POLSKO-JAPOŃSKA
WYŻSZA SZKOŁA
TECHNIK KOMPUTEROWYCH

Kamil Bilczyński
Wojciech Pazdur
Przemysław Pomorski
Jarosław Zieliński

Interaktywna grafika

WYDAWNICTWO PJWSTK

Kamil Bilczyński –pracuje jako dyrektor artystyczny w firmie The Farm 51, jest projektantem i twórcą scenerii do gier komputerowych. Autor artykułów i szkoleń poświęconych tworzeniu grafiki do gier komputerowych w prasie fachowej i Internecie. Cechuje go bogata wiedza z zakresu architektury i modelowania przestrzennego, znajomość narzędzi graficznych oraz specjalizowanych edytorów służących do budowania scenerii w grach komputerowych.

Wojciech Pazdur –jest absolwentem wydziału Automatyki, Elektroniki i Informatyki Politechniki Śląskiej. Specjalizuje się w projektowaniu gier wideo oraz tworzeniu grafiki komputerowej. Przez wiele lat współpracował z Wydawnictwem Helion, tłumacząc i pisząc książki związane z grafiką komputerową, był redaktorem naczelnym Magazynu 3D. Obecnie jest dyrektorem produkcji w firmie The Farm 51, produkującej gry komputerowe oraz wykładowcą Polsko-Japońskiej Wyższej Szkoły Technik Komputerowych.

Przemysław Pomorski –z wykształcenia jest specjalistą od tworzenia fabuł i narracji filmowej. Autor opowiadań, scenariuszy i projektant gier, obecnie zatrudniony w firmie The Farm 51 jako kierownik działu rozgrywki. Wieloletni pedagog, znawca militariów i miłośnik sztuk walki.

Jarosław Zieliński –jest ekspertem z dziedziny animacji postaci. Jako kierownik działu animacji w firmie The Farm 51 pracował nad wieloma projektami gier o zróżnicowanej tematyce, współtworząc rozwiązania technologiczne związane z animowanie postaci ludzkich i zwierzęcych. Specjalizuje się w wykorzystaniu technologii motion capture w powiązaniu z programami MotionBuilder i Maya.

Streszczenie

Z grafiką interaktywną (real-time) w ostatnich latach mamy do czynienia niemal wszędzie – od gier na telefony komórkowe, komputery osobiste i konsole, przez wizualizacje naukowe i reklamowe w kioskach prezentacyjnych lub przeglądarkach internetowych, po nawigację samochodową bądź animowane menu nowoczesnego telewizora. Niniejsza książka przedstawia przegląd i wstęp do różnych technik pracy dotyczących tworzenia trójwymiarowej grafiki interaktywnej głównie pod kątem gier akcji, jako najbardziej zaawansowanej dziedzinie z tego obszaru. W grach akcji zazwyczaj mamy do czynienia z dążeniem do maksymalnego fotorealizmu, co w połączeniu z dynamiką animacji i efektów specjalnych stawia największe wymagania twórcom zarówno rozwiązań technologicznych jak i elementów grafiki wykorzystywanych w takich projektach. Większość zagadnień omówionych w kolejnych rozdziałach książki można przenieść na grunt dowolnej aplikacji trójwymiarowej, która ma mieć charakter interaktywny lub w której grafika ma być renderowana w czasie rzeczywistym (ang. real-time). Głównym celem przedstawianych tu problemów ma być przybliżenie czytelnikowi podstawowych założeń, metod pracy i narzędzi służących do tworzenia różnych elementów grafiki real-time.

Seria: Podręczniki akademickie

Edytor serii: Leonard Bolc
Tom serii: 61

POLSKO-JAPOŃSKA
WYŻSZA SZKOŁA
TECHNIK KOMPUTEROWYCH

Kamil Bilczyński
Wojciech Pazdur
Przemysław Pomorski
Jarosław Zieliński

Interaktywna grafika

WYDAWNICTWO PJWSTK

Edytor
Leonard Bolc

Kierownik projektu
Prof. dr hab. inż. Konrad Wojciechowski

Korekta
Anna Bittner

Redaktor techniczny
Aneta Ługowska

Komputerowy skład tekstu
Grażyna Domańska–Żurek

Projekt okładki
Rafał Masłyk

Wydawnictwo
Polsko-Japońskiej Wyższej Szkoły Technik Komputerowych
ul. Koszykowa 86, 02-008 Warszawa
tel. 22 58 44 526, fax 22 58 44 503
e-mail: oficyna@pjwstk.edu.pl

Oprawa miękka
ISBN 978-83-63103-15-6
nakład: 150 egz.

Wersja elektroniczna
ISBN 978-83-63103-63-7

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Projekt „ Nowoczesna kadra dla e-gospodarki – program rozwoju Wydziału Zamiejscowego Informatyki w Bytomiu Polsko- Japońskiej Wyższej Szkoły Technik Komputerowych. Współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego. Poddziałanie 4.1.1 „ Wzmocnienie potencjału dydaktycznego uczelni" Programu Operacyjnego Kapitał Ludzki

This book should be cited as:

Bilczyński, K. et al., 2012. Interaktywna grafika. Warszawa: Wydawnictwo PJWSTK.

Spis treści

1

Wstęp do interaktywnej grafiki 3D

1.1 Wprowadzenie

Terminu „grafika interaktywna" używamy głównie w sytuacjach, gdy urządzenie elektroniczne wyposażone w wyświetlacz daje nam możliwość interakcji z interfejsem aplikacji i w wyniku tej interakcji obrazy na ekranie podlegają pewnym zmianom. Oznacza to, że z grafiką interaktywną w ostatnich latach mamy do czynienia niemal wszędzie - od gier na telefony komórkowe, komputery osobiste i konsole, poprzez wizualizacje naukowe i reklamowe w kioskach prezentacyjnych lub przeglądarkach internetowych, po nawigację samochodową bądź animowane menu nowoczesnego telewizora. Czy w grę wchodzi zmieniający się krajobraz w symulatorze lotu (rysunek 1.1), czy dynamiczna walka z potworami w grze akcji (rysunek 1.2), czy trójwymiarowa mapa w podręcznym urządzeniu do nawigacji (rysunek 1.3), gdy wpływamy na efekty wizualne wyświetlane przez program, wtedy mówimy o grafice interaktywnej.

Grafika interaktywna jest to jednocześnie dziedzina, która już w obrębie podstawowych pojęć sprawia pewne kłopoty. Niektóre z powyższych zastosowań (np. wizualizacje naukowe) wcale nie muszą opierać się na interakcji z użytkownikiem, a jednak ze względu na narzędzia i metody wykorzystane do ich tworzenia, ze względów technologicznych praktycznie nie ma różnicy pomiędzy interaktywną a nieinteraktywną wersją danej aplikacji. Wtedy powinniśmy mówić raczej o grafice czasu rzeczywistego (lub grafice generowanej w czasie rzeczywistym), z tym że tutaj znowu „czas rzeczywisty" zasadniczo nie odnosi się do samego procesu tworzenia grafiki, a jedynie do przetwarzania obrazu w sposób umożliwiający wyświetlenie animacji zmienianej przez procesor graficzny z szybkością kilkunastu lub kilkudziesięciu klatek na sekundę. Pojawiają się jednak aplikacje, w których grafika faktycznie jest generowana w czasie rzeczywistym (np. tło gry muzycznej generowane w oparciu o ścieżkę dźwiękową) i do tego w powiązaniu z interaktywnym charakterem programu. W języku angielskim bardzo często stosuje się określenie całości tych zagadnień jako grafiki czasu rzeczywistego (real-time).

Rysunek 1.1. W symulatorze lotu nasze działania wpływają nie tylko na wygląd krajobrazu widocznego z kabiny samolotu, ale także na wskaźniki i ekrany kokpitu - pierwszy z tych elementów jest zazwyczaj trójwymiarowy, drugi może być, w zależności od przyjętych rozwiązań, wyświetlany jako grafika 2D lub 3D. W obu przypadkach mamy do czynienia z grafiką interaktywną. Widok z jednej ze starszych wersji gry „Microsoft Flight Simulator"

Rysunek 1.2. Trójwymiarowa gra akcji wymaga animowania bardzo dużej liczby elementów w sposób interaktywny (krajobraz, postacie, rekwizyty, pojazdy). Widok z gry „NecroVisioN: Lost Company"

Rysunek 1.3. W przypadku nawigacji samochodowej interaktywność grafiki jest pozornie niewielka, jednak sposób wyświetlania mapy i wskaźników informacyjnych zależy nie tylko od operacji na interfejsie programu, ale też od poruszania się użytkownika względem satelity. Widok urządzenia nawigacyjnego firmy TomTom

Na zagadnienia dotyczące grafiki interaktywnej, podobnie jak na wszelkie inne aspekty tworzenia grafiki komputerowej, możemy również patrzeć pod kątem tego, czy jest ona dwu- czy trójwymiarowa. Interaktywna grafika dwuwymiarowa to przede wszystkim same interfejsy różnych aplikacji użytkowych oraz urządzeń codziennego użytku, które coraz częściej mają postać wyświetlacza LCD z wirtualnymi, animowanymi przyciskami i efektami towarzyszącymi ich użyciu. Oprócz tego z grafiką dwuwymiarową w ujęciu interaktywnym stykamy się w grach komputerowych i wideo, telefonach komórkowych czy nawet kasownikach autobusowych. Kwestie tworzenia interaktywnej grafiki dwuwymiarowej zostały już stosunkowo dobrze ustandaryzowane od strony technologicznej i zazwyczaj zaprojektowanie oraz wykonanie danego interfejsu bądź wizualizacji wymaga głównie odpowiedniej wiedzy z zakresu ergonomii, estetyki, technologii i specyficznych dziedzin, których dotyczy dane zagadnienie. Dzięki narzędziom programistycznym na różne platformy oraz gotowym aplikacjom takim jak np. Flash, w grafice dwuwymiarowej nie ma obecnie z roku na rok zbyt wielkich przeobrażeń dotyczących narzędzi, metod pracy, koncepcji technolo-

gicznych czy też kosztów realizacji. Zupełnie inaczej wygląda na tym tle tworzenie grafiki trójwymiarowej - tutaj ciągle jeszcze nie osiągnięto pułapu, który można by określić jako bliski maksimum możliwości człowieka i technologii, a co za tym idzie, każdego roku pojawiają się nowe narzędzia oraz technologie, które pozwalają albo wykonać pewne zadania dużo szybciej, albo podnieść jakość efektu końcowego, albo też mocno obniżyć koszty udostępnienia danych rozwiązań szerokiemu odbiorcy.

Niniejsza książka przedstawia przegląd oraz wstęp do różnych technik pracy dotyczących tworzenia trójwymiarowej grafiki interaktywnej, głównie pod kątem gier akcji, jako najbardziej zaawansowanej dziedzinie z tego obszaru (rysunek 1.4). W grach akcji zazwyczaj mamy do czynienia z dążeniem do maksymalnego fotorealizmu, co w połączeniu z dynamiką animacji i efektów specjalnych stawia największe wymagania twórcom zarówno rozwiązań technologicznych jak i elementów grafiki wykorzystywanych w takich projektach.

Rysunek 1.4. Gry akcji z animacją skomplikowanych modeli postaci i otoczenia w czasie rzeczywistym stanowią największe wyzwanie dla grafików i programistów. Widok z gry „Two Worlds II"

Większość zagadnień omówionych w kolejnych rozdziałach książki można przenieść na grunt dowolnej aplikacji trójwymiarowej, która ma mieć charakter interaktywny albo w której grafika ma być renderowana w czasie rzeczywistym (ang. *real-time*). Głównym celem przedstawianych tu problemów ma być przy-

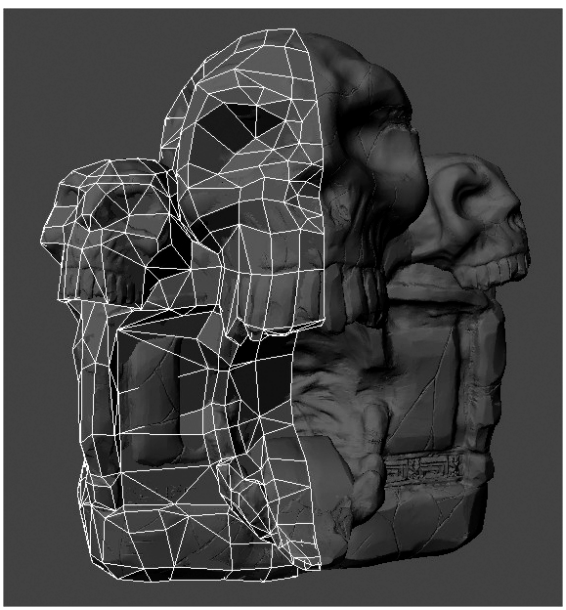
bliżenie czytelnikowi podstawowych założeń, metod pracy i narzędzi służących do tworzenia różnych elementów grafiki real-time.

1.2 Podstawowe ograniczenia i problemy

Sposoby tworzenia elementów grafiki trójwymiarowej mogą się diametralnie różnić w zależności od projektu, platformy sprzętowej, wybranej technologii programistycznej i wielu innych czynników. W kolejnych rozdziałach książki przedstawimy techniki i narzędzia stosowane w różnych rzeczywistych sytuacjach, natomiast większość omówionych rozwiązań praktycznych opiera się na najbardziej uniwersalnych metodach tworzenia grafiki interaktywnej.

Głównymi założeniami elementów grafiki 3D tworzonych na potrzeby do gier czy innych aplikacji interaktywnych są:

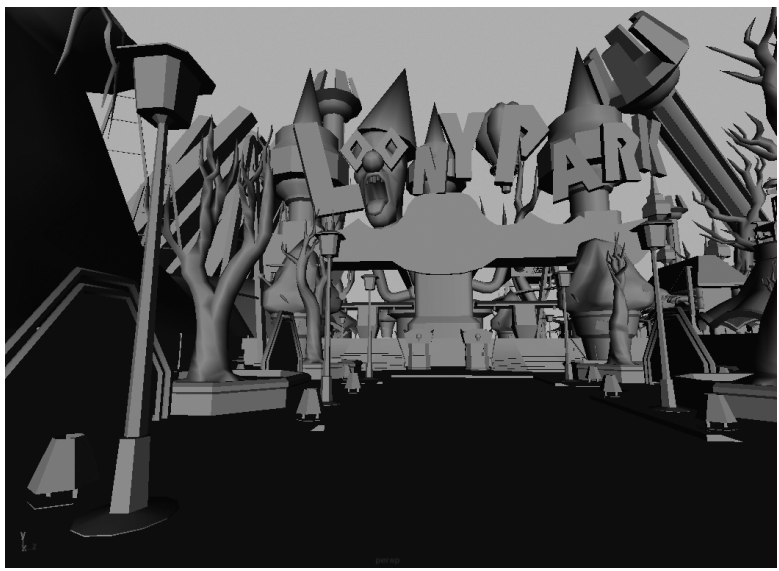
- Prosta struktura modeli (mała ilość danych na jeden model) umożliwiająca równoczesne przechowywanie wielu obiektów w pamięci, efektywne zarządzanie nimi przez aplikację i płynne odświeżanie sceny wiele razy na sekundę (rysunek 1.5). Obiekty tworzone bezpośrednio na potrzeby grafiki interaktywnej nazywamy z tego powodu niskopoligonowymi (ang. *low-poly*), czyli zawierającymi małą liczbę ścianek (ang. *polygons*).



Rysunek 1.5. Siatka low-poly - kamiennej rzeźby z czaszkami jest stosunkowo prostym modelem, jego szczegółowy wygląd uzyskuje się za pomocą odpowiednio dobranego materiału i tekstury. Model z gry „Adventurer"

- Wykorzystanie modeli siatkowych (bryły zbudowane z wielokątów) pokrytych teksturami w postaci bitmap jako najprostszej obliczeniowo reprezentacji skomplikowanych obiektów trójwymiarowych (rysunek 1.6a i 1.6b).

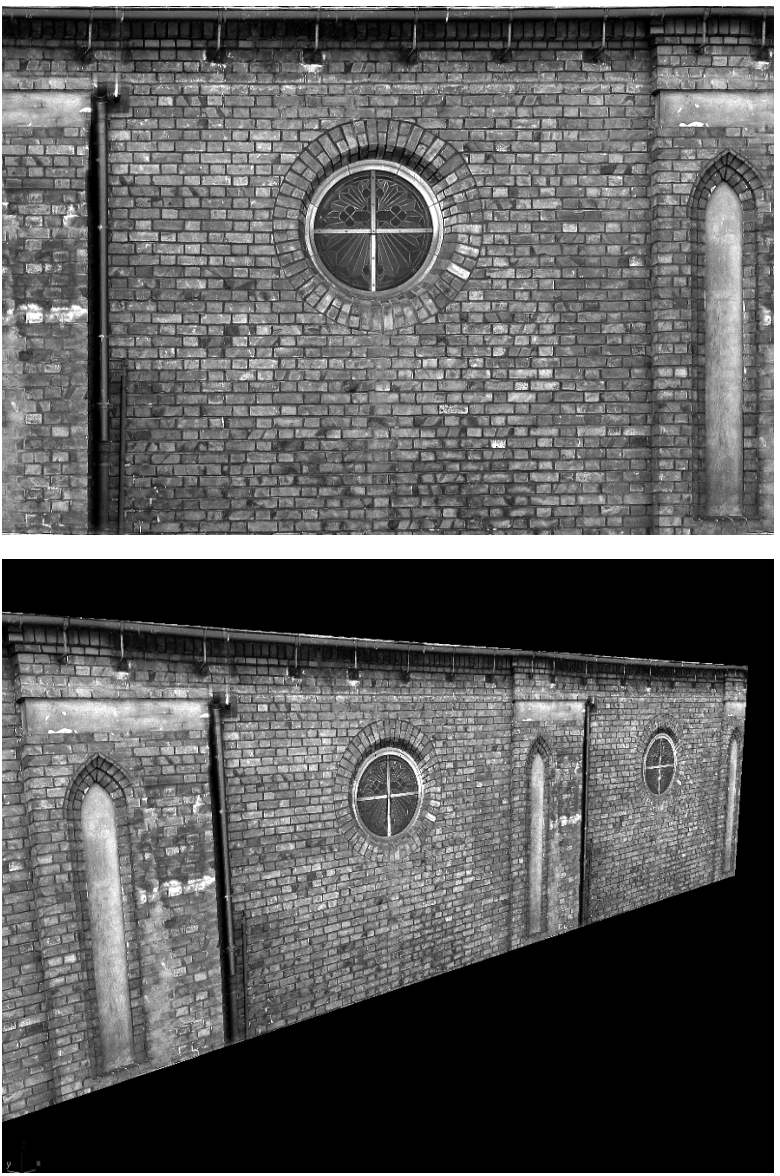
a)



b)

Rysunek 1.6. Nawet bardzo skomplikowany krajobraz można zbudować z bardzo prostych brył low-poly (obraz u góry). Po nałożeniu tekstur i oświetleniu scena staje się szczegółowa i fotorealistyczna (obraz u dołu). Widoki z gry „Painkiller"

- Możliwość powtarzalnego wykorzystania tych samych elementów (np. do zbudowania lasu wykorzystujemy tylko kilka unikalnych modeli drzew i kilka tekstur liści, do pokrycia długiej ściany korzystamy z zapętlonej tekstury - rysunek 1.7);



Rysunek 1.7. U góry - specjalnie przygotowana tekstura, którą można wykorzystać do stworzenia powtarzalnego fragmentu dużej ściany (u dołu)

- Dobór stopnia skomplikowania obiektu i rozdzielczości tekstur w zależności od odległości od kamery, kąta obserwacji i roli odgrywanej w scenie.
- Wykorzystanie płaskich powierzchni z nałożonymi teksturami jako elementów tła lub składników bardziej złożonych struktur (np. dym lub krople wody symulujemy zbiorem cząsteczek będących półprzezroczystymi prostokątami).
- Stosowanie specyficznych technik oświetleniowych i materiałów dla powierzchni obiektów, które są zoptymalizowane pod kątem renderingu w czasie rzeczywistym.
- Używanie tylko tych efektów specjalnych, których generowanie jest możliwe w sposób płynny z częstotliwością kilkudziesięciu razy na sekundę.

Ograniczenia te są narzucone głównie wydajnością renderingu poszczególnych platform sprzętowych (komputery PC, konsole, urządzenia przenośne). Przy różnych urządzeniach mamy więc do czynienia z zupełnie innymi wymaganiami (pod względem ilościowym), jednak zasadnicze założenia, które mają na celu zmaksymalizować jakość i płynność wyświetlania grafiki, stosuje się w identyczny sposób dla każdej platformy.

W praktyce z powyższych ograniczeń wynika, że obiekty i sceny do grafiki interaktywnej przygotowywane są w następujący sposób:

- Scena może zawierać naraz od kilkudziesięciu do kilkuset obiektów, wliczając w to małe elementy tła, takie jak kamyki na podłożu czy rośliny. Warto jednak pamiętać, że część elementów może być zgrupowana w jednym obiekcie (np. kilka budynków na drugim planie może fizycznie stanowić jeden obiekt).
- Wszystkie obiekty, które mogą być powtarzalne (np. pojazdy tej samej marki), tworzymy tylko jeden raz i kopiujemy, w niektórych przypadkach stosując specyficzne zabiegi w celu uniknięcia efektu powtarzalności (np. każdy pojazd ma dodatkową płaszczyznę w celu stworzenia innej tablicy rejestracyjnej, każdy żołnierz w scenie ma unikalny model głowy chociaż mundury mogą być identyczne itd.).
- Liczba tekstur wykorzystanych w scenie również zazwyczaj nie przekracza kilkudziesięciu lub kilkuset plików - ze względu na ilość pamięci zajmowanej przez pliki graficzne ale także z powodu ograniczania liczby unikalnych materiałów używanych przez obiekty.

1.3 Szczegółowość obiektów i wykorzystanie tekstur

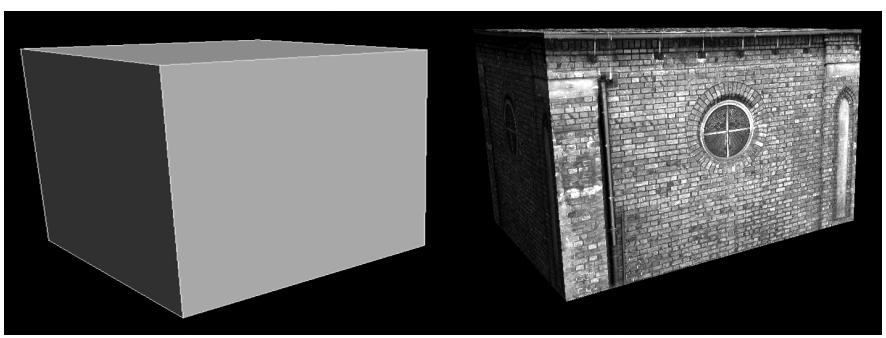
Podsumowując zebrane wyżej uwagi, można stwierdzić, że z punktu widzenia grafika, głównym problemem do rozwiązania jest kwestia tego, w jaki sposób tworzyć obiekty, które są bardzo proste, jednak sprawiają wrażenie odpowiednio skomplikowanych. Najczęściej mamy do czynienia z narzuconym z góry ograniczeniem odnośnie złożoności modelu, np. model pojazdu powinien mieć nie

więcej niż 3 tysiące ścianek i składać się z 5 siatek (nadwozie plus koła), a model postaci nie więcej niż 8 tysięcy ścianek i składać się z dwóch siatek (korpus z kończynami i głowa). Do tego zazwyczaj dochodzą limity liczby i rozmiarów tekstur (np. nie więcej niż 2 tekstury w rozdzielczości 1024x1024 dla pojedynczej postaci).

Przy tak formułowanych ograniczeniach nie mamy zbyt wielkiej możliwości operowania poziomem szczegółowości modelu, dlatego pozostaje nam przyjęcie odpowiednich technik generowania jak najbardziej złożonych tekstur i materiałów. Oznacza to, że dla danego silnika graficznego (systemu wyświetlania grafiki) dobieramy możliwie najlepszy sposób uszczegółowienia modelu za pomocą tekstur.

1.3.1 Tekstury koloru

Domyślnie praktycznie każdy silnik graficzny umożliwia nałożenie na siatkę tekstury koloru (ang. *diffuse texture*), która określa barwę poszczególnych punktów powierzchni w oparciu o załadowany do pamięci obraz bitmapowy (rysunek 1.8).



Rysunek 1.8. Najbardziej podstawową techniką uszczegóławiania obiektów real-time jest nakładanie na nie tekstury koloru, tą metodą nawet z bardzo prostej bryły niskopoligonowej można uzyskać powierzchnię o dużej szczegółowości wizualnej

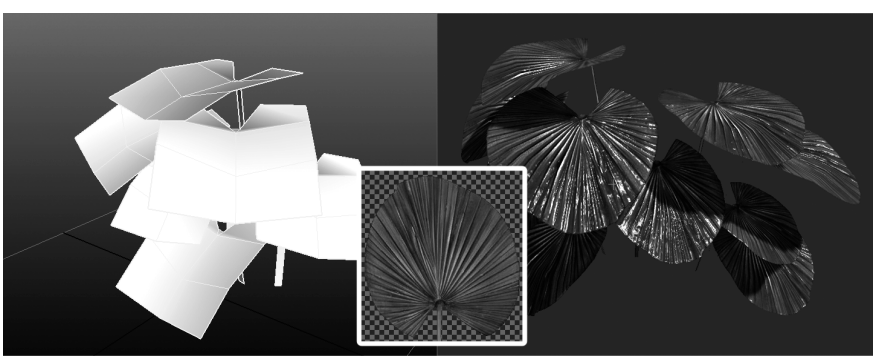
1.3.2 Tekstury odblasków

Oprócz tekstury koloru, różne silniki 3D umożliwiają dodawanie dodatkowych efektów opartych na teksturach. Dla uzyskania szczegółowości niektórych powierzchni ważne może być wykorzystanie np. tekstur odblasków (ang. *specular*), które określają, w których miejscach obiekt jest bardziej lub mniej błyszczący, co daje efekt dużo lepszej interakcji powierzchni ze światłem i symuluje większą złożoność mikrostruktur obiektu (rysunek 1.9).

Rysunek 1.9. Mapa odblasków widoczna po prawej stronie rysunku powoduje, że płaska powierzchnia modelu błyszczy się w odpowiedni sposób, imitując skomplikowaną strukturę jej materiału. Widok z gry „Painkiller"

1.3.3 Tekstury przezroczystości

Wielu kształtów z uwagi na ich naturalną złożoność nie opłaca się w ogóle modelować za pomocą siatek. Dotyczy to między innymi roślin, których liście doskonale można zasymulować za pomocą płaskich powierzchni z teksturą. W tym przypadku niemal zawsze dodajemy do tekstury maskę przezroczystości, która określa kształt liścia wycięty z prostokątnej powierzchni jak też pozwala ukryć te obszary powierzchni, które znajdują się poza powierzchnią liścia (rysunek 1.10). Umożliwia to tworzenie bardzo złożonych struktur roślinności przy bardzo prostych modelach i teksturach (rysunek 1.11).



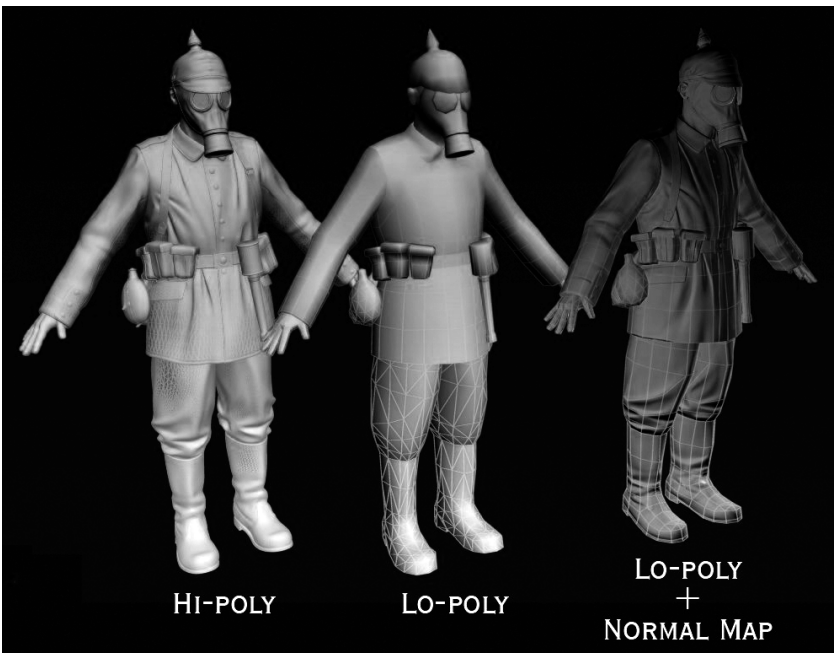
Rysunek 1.10. Model zbudowany z prostych powierzchni (po lewej) po nałożeniu na niego tekstury z maską przezroczystości (pośrodku) może dać efekt złożonej organicznej struktury (po prawej)

Rysunek 1.11. W większości gier roślinność symulowana jest za pomocą bardzo prostych obiektów z teksturami wykorzystującymi maski przezroczystości, takie jak model z rysunku 1.10

1.3.4 Tekstury (mapy) normalnych

Jedną z nader mocno rozwijanych i wykorzystywanych w ostatnich latach metod uszczegóławiania obiektów w grach jest użycie map normalnych (ang. *normal-maps*). Zagadnieniu temu poświęcono sporo miejsca w następnych rozdziałach.

Wykorzystanie map normalnych polega na przeniesieniu informacji o skomplikowanej strukturze modelu wysokopoligonowego (czyli bryły o zbyt dużej liczbie ścianek, by mogła być wykorzystana w aplikacji interaktywnej) na prostą bryłę niskopoligonową za pomocą specjalnie wygenerowanej tekstury. Tekstura ta, zwana mapą normalnych, ma zapisane w sobie informacje o wektorach normalnych siatki wysokopoligonowej, czyli o sposobie odbijania światła przez skomplikowany model. Po nałożeniu takiej tekstury na model low-poly uzyskujemy dużo większą wizualną złożoność modelu, ponieważ jego powierzchnia reaguje na światło w taki sposób, jakby była wielokrotnie bardziej skomplikowana. Technika ta wymaga jednak dość czasochłonnego modelowania, ponieważ oprócz modelu niskopoligonowego (low-poly) konieczne jest też stworzenie modelu wysokopoligonowego (hi-poly) aby wygenerować mapy normalnych. Jednak efekt końcowy zazwyczaj wart jest wysiłku, ponieważ model niskopoligonowy z mapą normalnych jest często wizualnie nie do odróżnienia w porównaniu z modelem dużo bardziej skomplikowanym a przy tym zajmuje stosunkowo niewiele miejsca w pamięci i można łatwo go animować.



Rysunek 1.12. Rysunek 1.12. Model postaci wykorzystujący mapy normalnych

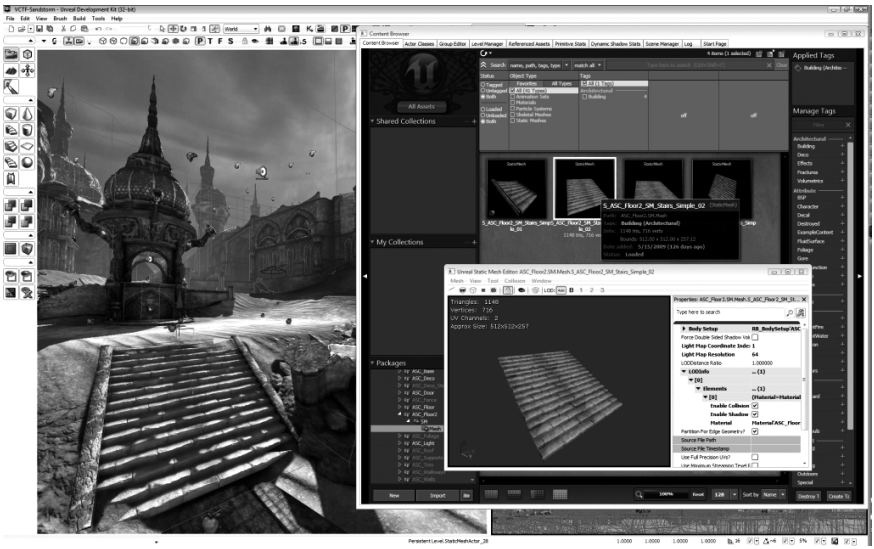
1.4 Podstawowe narzędzia do tworzenia grafiki interaktywnej

Podczas pracy nad elementami grafiki interaktywne przez większość czasu wykorzystuje się standardowe programy graficzne, wykorzystywane we wszelkich innych zastosowaniach grafiki 3D. W przykładach zaprezentowanych w niniejszej książce posługujemy się przede wszystkim pakietem Maya firmy Autodesk oraz programem Photoshop firmy Adobe. Oprócz tego wspominamy też o kilku dodatkowych narzędziach, z których można lub trzeba korzystać w określonych sytuacjach.

Co istotne, w opisach przedstawionych w książce posługujemy się na tyle uniwersalnymi narzędziami, że nawet wtedy, gdy dany przykład dotyczy programu Maya, można go wykonać bez większych problemów w innych pakietach, np. 3ds Max. W niektórych przypadkach nazwy pewnych funkcji brzmią inaczej, jednak ze względu na ograniczone ramy niniejszej książki musimy założyć, że czytelnik potrafi posługiwać się najważniejszymi poleceniami i modułami wybranego przez siebie programu. Do samodzielnego wykonania prac opisanych w dalszych rozdziałach niniejszej książki czytelnik musi posiadać podstawową wiedzę odnośnie modelowania i wykorzystania materiałów na obiektach 3D. W razie problemów zachęcamy do sięgnięcia po literaturę opisującą bardziej szczegółowo pracę z poszczególnymi aplikacjami.

Standardowo stanowisko pracy dla osoby tworzącej grafikę interaktywną wygląda następująco:

- Komputer PC z wydajną kartą graficzną (do podglądu scen w czasie rzeczywistym), dużą ilością pamięci RAM i mocnym procesorem (szczególnie gdy modelujemy bryły wysokopoligonowe lub renderujemy oświetlenie na złożonych scenach).
- Pakiet do grafiki 3D, taki jak 3ds Max, Maya, Blender itd. Z programu takiego korzystamy przede wszystkim do modelowania, mapowania tekstur i animacji. Czasem w programie 3D konfigurujemy też materiały przypisane do powierzchni obiektu, właściwości fizyczne modelu wykorzystywane później w symulacjach czasu rzeczywistego. Używany przez nas program musi posiadać opcje eksportu modeli do formatów obsługiwanych przez silnik graficzny danej aplikacji interaktywnej. W chwili obecnej większość silników 3D standardowo współpracuje z pakietami 3ds Max i Maya.
- Program do grafiki 2D (rastrowej), na przykład Adobe Photoshop lub Corel Photopaint. W programie tym tworzymy i obrabiamy tekstury i maski wykorzystywane w materiałach obiektu. W zależności od wykorzystanego silnika aplikacji interaktywnej, do programu 2D może być konieczne doinstalowanie zewnętrznych plug-inów, które służą do zapisu tekstur w odpowiednich formatach.
- Program do rzeźbienia brył wysokopoligonowych (hi-poly), jeżeli konieczne jest tworzenie map normalnych dla złożonych modeli. Wśród tego typu programów najpopularniejsze to Pixologic Zbrush oraz Autodesk Mudbox, choć w ograniczonym zakresie funkcje rzeźbienia dostępne są też standardowo w pakietach typu 3ds Max i Maya.
- Dedykowane programy lub plug-iny do innych programów związane z tworzeniem i obróbką map normalnych. W przypadku, kiedy nie chcemy tworzyć modeli wysokopoligonowych, uproszczone mapy normalnych możemy generować za pomocą specjalnych plug-inów Photoshopa (Nvidia Normal Map Filter, nDo) lub zewnętrznych aplikacji (Crazybump) w oparciu o tekstury koloru lub specjalne mapy wypukłości.
- Silniki aplikacji interaktywnych - aplikacja lub gra, nad którą mamy pracować, jest zawsze budowana w oparciu o określone środowisko programistyczne. Potocznie system służący do wyświetlania grafiki interaktywnej i budowania aplikacji tego typu nazywamy silnikiem (ang. *engine*). Obecnie na rynku dostępnych jest wiele silników, zarówno darmowych jak też i płatnych, na których można oprzeć swoje aplikacje. W niniejszej książce przytaczamy przykłady wykorzystania silnika Unreal Technology firmy Epic Games, który jest o tyle dobrym systemem szkoleniowym, że poza możliwością obejrzenia i przeanalizowania wielu znakomicie zrealizowanych gier stworzonych przy jego użyciu (Unreal Tournament, Gears of War), technologia ta posiada bogatą dokumentację i jest dostępna w wersji darmowej do niekomercyjnych zastosowań (pod postacią Unreal Development Kit, w skrócie UDK, www.udk.com). Poza UDK warte szczególnego zwrócenia



Rysunek 1.13. Przykładowa lokacja gry 3D w silniku Unreal Technology i narzędzia do zarządzania obiektami w projekcie (materiały firmy Epic Games)

uwagi są silniki Unity 3 firmy Unity Technologies (www.unity3d.com) oraz CryENGINE firmy CryTek (www.mycryengine.com), które również pozwalają na przetestowanie rozbudowanego systemu do tworzenia gier i aplikacji interaktywnych. Prace zrealizowane po przeczytaniu tej książki można z powodzeniem wykorzystać w każdej z wymienionych technologii.

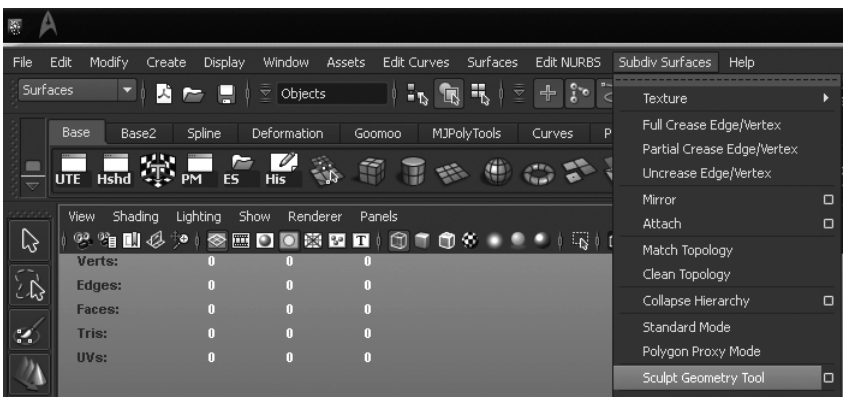
2

Uszczegóławianie modeli wysokopoligonowych

Podczas pracy nad modelami wysokopoligonowymi często zachodzi potrzeba wzbogacenia naszych siatek dodatkowymi szczegółami, fakturami i wygładzenia ich powierzchni lub ostrych krawędzi. Aby tego dokonać musimy skorzystać z dodatkowych narzędzi pakietu Maya lub zewnętrznych wtyczek i programów. W poniższym rozdziale przybliżymy kilka najważniejszych technik, które służą do uszczegóławiania modeli.

2.1 Narzędzia rzeźbiarskie (Sculpt Geometry Tool, Zbrush)

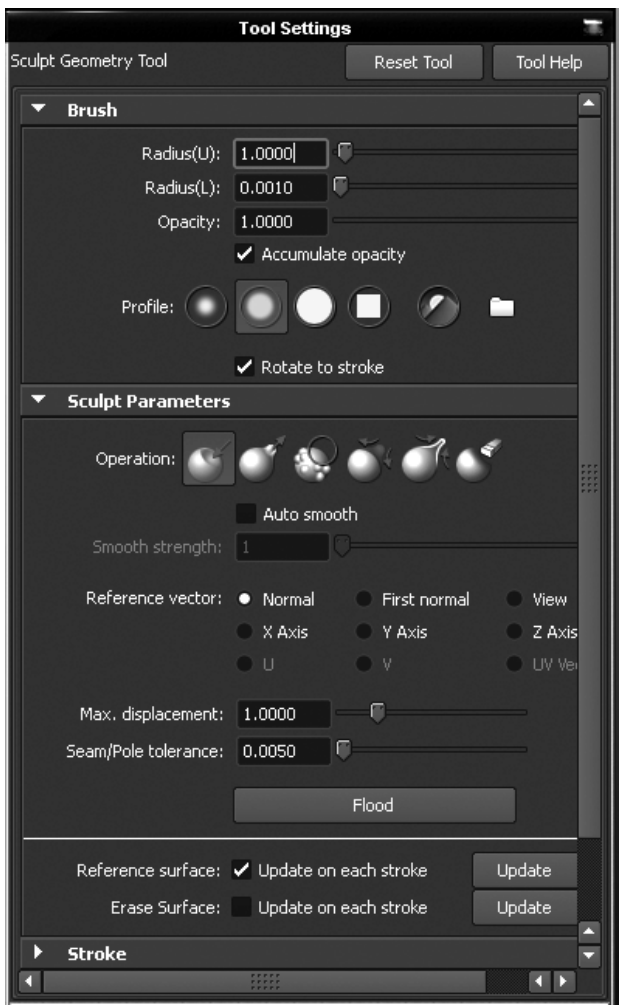
Pierwszą z technik uszczegóławiania obiektów jest wykorzystanie funkcji *Sculpt Geometry Tool* programu Maya - jest to narzędzie będące substytutem obróbki rzeźbiarskiej ukierunkowane na edycje siatek organicznych. To właśnie przy użyciu narzędzia *Sculpt* możemy wyrzeźbić nierówności terenu oraz wszel-



Rysunek 2.1. Wybór funkcji Sculpt Geometry Tool

kiego typu organiczne siatki, takie jak mięśnie, twarze itp., czyli wszystko to, gdzie tradycyjna edycja ściankowa jest nieskuteczna lub mało efektywna. Często też dzięki wykorzystaniu narzędzi Sculpt możemy zaoszczędzić sporo czasu, ponieważ pewne zadania wykonamy dużo efektywniej niż przy użyciu standardowego wytłaczania, deformowania itd.

Aby przejść do edycji siatek przez narzędzie *Sculpt*, musimy z górnego menu wybrać polecenie *Subdiv Surface/Sculpt Geometry Tool* (rysunek 2.1). Należy pamiętać, że operacja rzeźbienia narzędziem *Sculpt* ma sens tylko w przypadku siatek o dużej liczbie ścianek, gdyż wszelkie szczegóły będą miały rozdzielczość (dokładność) jedynie taką jaką ma liczba ścianek w ich obrębie.



Rysunek 2.2. Ustawienia narzędzia Sculpt Geometry Tool

W razie potrzeby zagęszczenia siatki możemy posłużyć się narzędziem *Mesh/ Smooth*.

Kiedy korzystamy z narzędzia *Sculpt Geometry Tool*, otwierane jest okno, w którym możemy ustawić dodatkowe właściwości „pędzla" rzeźbiarskiego (rysunek 2.2). (*Uwaga*: z punktu widzenia językowego powinniśmy mówić raczej o „dłucie" do rzeźbienia, ale w programach 3D narzędzia tego typu domyślnie nazywane są pędzlami - „brushes"). Opcje podzielone są na właściwości pędzla i właściwości rzeźbiarskie.

Do najważniejszych opcji pędzla (*Brush*) należą:

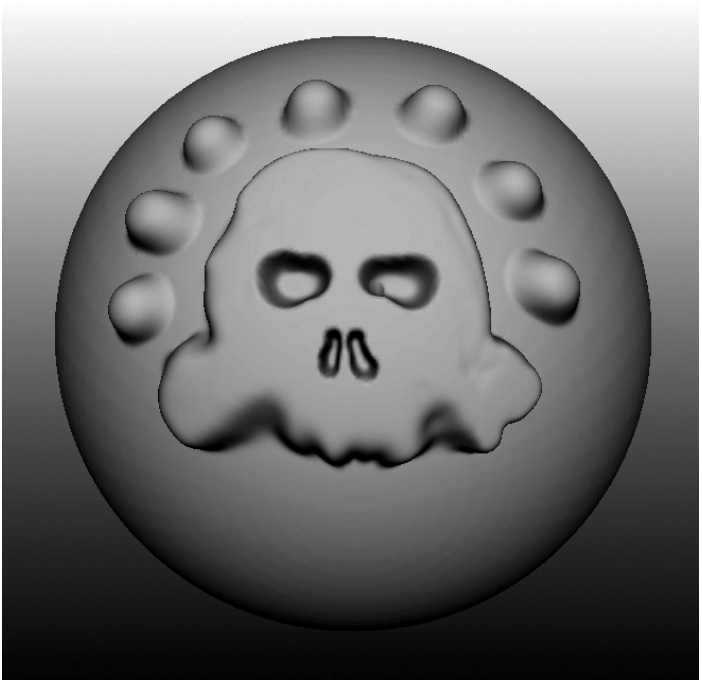
- *Radius(U)* i *Radius(L)* - zasięg i wielkość;
- *Opacity* - krycie powierzchni (odwrotność przezroczystości);
- *Profile* - różne kształty pędzli, możemy także załadować swój własny profil pędzla;
- *Rotate to stroke* - obracanie pędzla względem rzeźbionej powierzchni w trakcie przeciągania nim podczas rzeźbienia.

Do najważniejszych opcji rzeźbienia (*Sculpt*) należą:

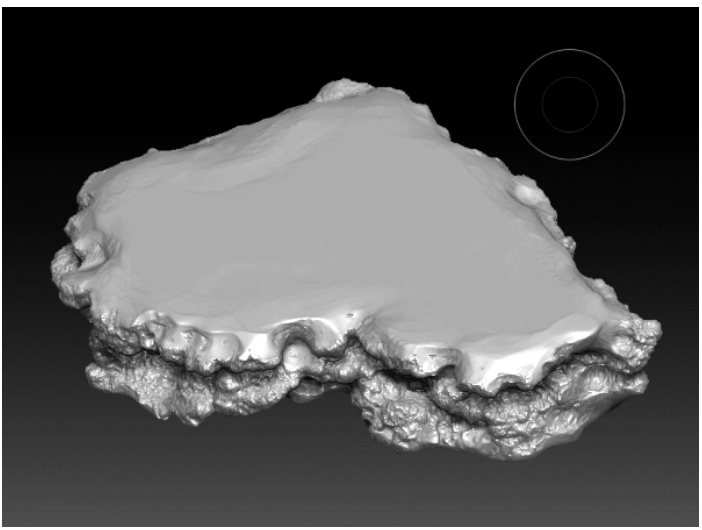
- *Operation* - sposób rzeźbienia, to tutaj wybieramy czy pędzel ma odkształcać siatkę robiąc wgłębienia czy wypukłości, możemy też wyciągać powierzchnie w górę lub je wygładzać itd.;
- *Auto smooth* - wygładzanie powierzchni;
- *Reference vector* - określa kierunek działania pędzla względem wybranej osi;

Po ustawieniu opcji rzeźbienia możemy malować pędzlem na powierzchni obiektu, zmieniając jego kształt. Przykład szybkiego użycia techniki *Sculpt* na kuli (*Sphere*) pokazuje rysunek 2.3. Jak widać, efekt przypomina bardziej ręczną zabawę z gliną niż obiekty tworzone tradycyjnymi technikami do modelowania ściankowego w programie Maya. Ta metoda dodawania szczegółów daje nam duże pole manewru gdy zachodzi potrzeba stworzenia modelu lub elementu modelu który przypominałby efekt zastosowania klasycznych technik rzeźbiarskich.

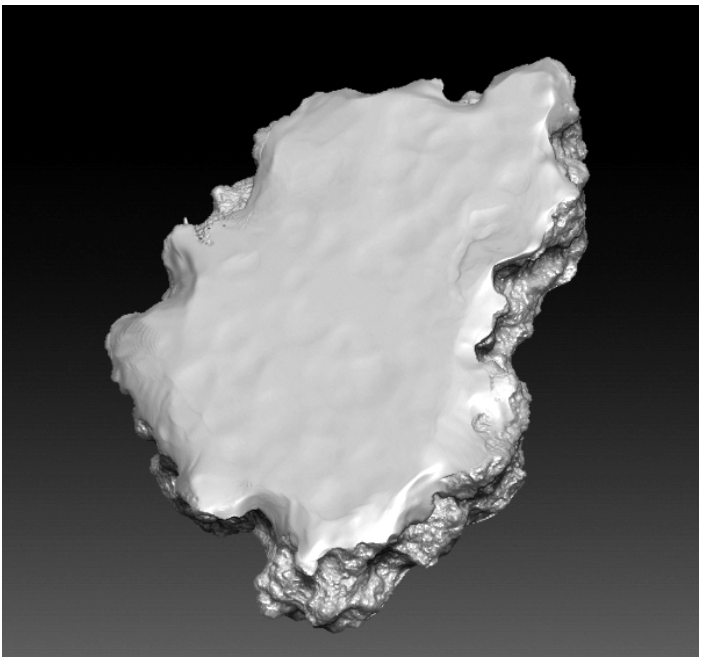
Rzeźbienie narzędziem *Sculpt* nie jest optymalną metodą tworzenia skomplikowanych form organicznych. Jeśli musimy tworzyć naprawdę złożone bryły o nieregularnych kształtach, należy zainteresować się dedykowanymi programami do rzeźbienia (sculptingu), takimi jak Autodesk Mudbox lub Pixologic ZBrush. Ich narzędzia dają podobną funkcjonalność jak pędzle rzeźbiarskie programu Maya, jednak są o wiele bardziej rozbudowane. Chociaż metodyka pracy z tymi programami wymaga poświęcenia dodatkowego czasu na wdrożenie się w szereg nowych zagadnień, praca z tego typu programami pozwala na o wiele bardziej efektywne i dokładne rzeźbienie skomplikowanych modeli (rysunki 2.4 i 2.5).



Rysunek 2.3. Przykład wykorzystania Sculpt Geometry Tool na sferze o dużym zagęszczeniu siatki



Rysunek 2.4. Przykład bryły wyrzeźbionej z programie Zbrush



Rysunek 2.5. Przykład bryły wyrzeźbionej z programie Zbrush

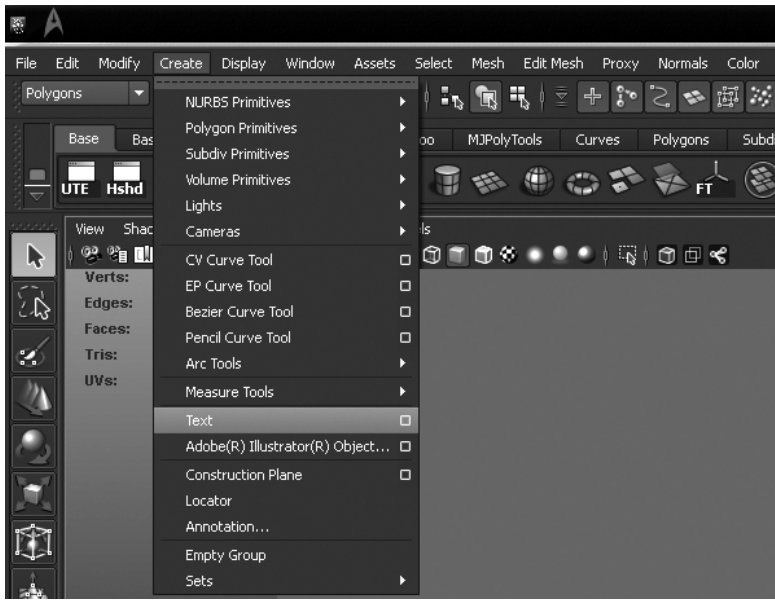
2.2 Dodawanie trójwymiarowych napisów (Text)

Kolejną techniką często wykorzystywaną do uszczegóławiania siatek jest dodawanie wytłoczonych lub wtłoczonych napisów. Aby dodać trójwymiarowy napis do sceny musimy z górnej listwy menu wybrać polecenie *Create/Text* (rysunek 2.6).

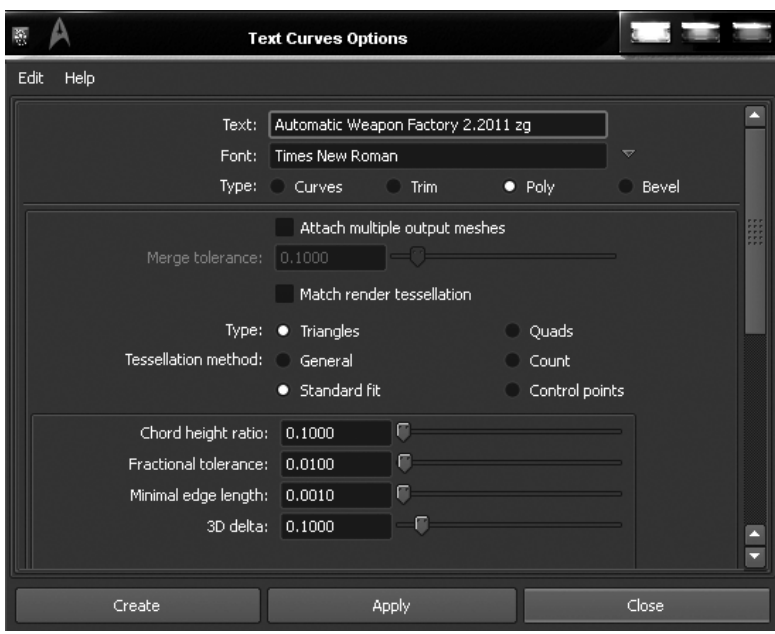
W oknie edycji możemy wprowadzić napis jaki ma zostać wymodelowany jak również ustawić podstawowe opcje jego tworzenia (rysunek 2.7).

Najważniejsze właściwości tworzenia tekstu:

- *Text* - tutaj wpisujemy treść napisu;
- *Font* - tutaj wybieramy czcionkę;
- *Type* - tu wybieramy za pomocą pomocy jakiej techniki zostanie wymodelowany tekst: czy będzie zbudowany z krzywych (*Curves*) czy tez ze ścianek (*Poly*);
- *Attach multiple output meshes* - tutaj możemy określić zakres łączenia siatek ze sobą;
- *Type* - w tym polu wybieramy, czy tekst ma być stworzony z trójkątnych ścianek (*Triangles*) czy czworokątów (*Quads*);



Rysunek 2.6. Wybór funkcji Create/Text



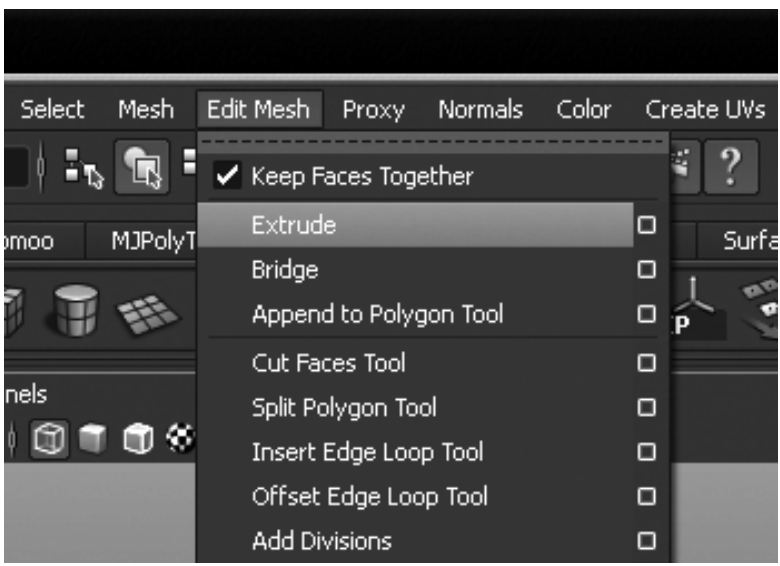
Rysunek 2.7. Menu Text Curves Options

- *Tesselation method* - określamy tu sposób podziału wymodelowanych ścianek, jeśli tworzymy obiekty typu *Poly* to najlepszą metodą podziału będzie *Standard fit*.

Po określeniu powyższych ustawień wystarczy w dole okna wcisnąć przycisk *Create* i na ekranie pojawi się wymodelowany napis w postaci modelu ściankowego (rysunek 2.8).

Rysunek 2.8. Podgląd tekstu stworzonego przy użyciu narzędzia Text

Ponieważ napis jest płaską bryłą, żeby nadać mu głębię musimy go wytłoczyć. W tym celu należy z górnej listwy menu wybrać polecenie *Edit Mesh/Extrude* (rysunek 2.9).



Rysunek 2.9. Operacja wytłaczania

Na rysunku 2.10 widzimy wygląd tekstu po operacji wtłaczania. Tak wytłoczony napis możemy dołączyć do bryły, na której powierzchni chcemy uzyskać tego typu efekt.

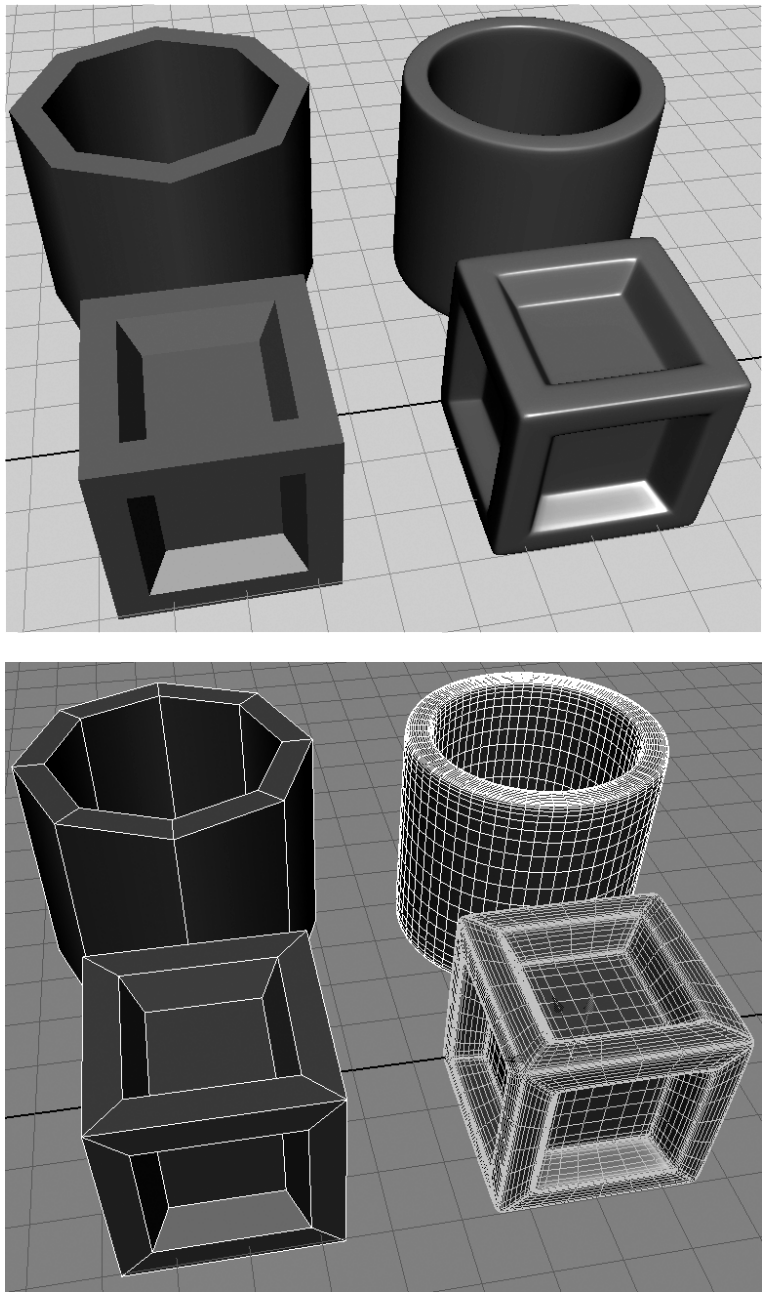
Rysunek 2.10. Finalny wygląd wytłoczonego napisu w widoku perspektywicznym

2.3 Zagęszczanie siatki

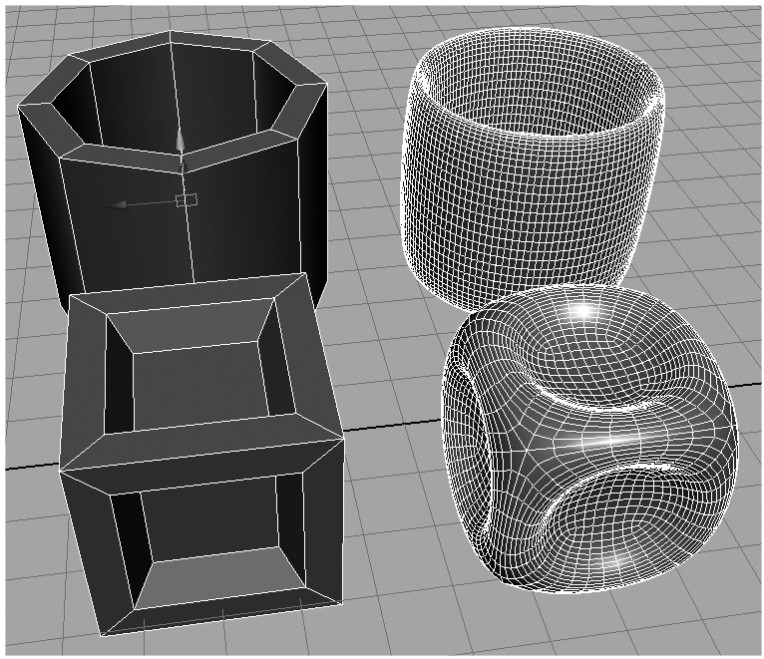
Jedną z najpopularniejszych metod modelowania obiektów wysokopoligonowych jest zbudowanie stosunkowo prostej bryły w postaci modelu ściankowego oraz poddanie tej bryły operacji zagęszczania, w wyniku której wygładzane są krawędzie i krzywizny powierzchni. Na rysunku 2.11 przedstawiono dwie bryły w wersji niskopoligonowej i po zagęszczeniu. Proste siatki ułatwiają edycję bryły oraz kontrolę nad wszystkimi jej elementami i można je wyświetlać bezpośrednio w silniku aplikacji interaktywnej, natomiast siatki zagęszczone zazwyczaj wykorzystywane są do generowania normal-map, o których więcej można przeczytać w rozdziale 3. niniejszej książki.

Narzędzia służące do automatycznego zagęszczania siatek pozwalają jednym kliknięciem wygenerować bryłę o gładkiej powierzchni, jednak by uzyskać pożądany kształt musimy zazwyczaj odpowiednio przygotować strukturę siatki uproszczonej.

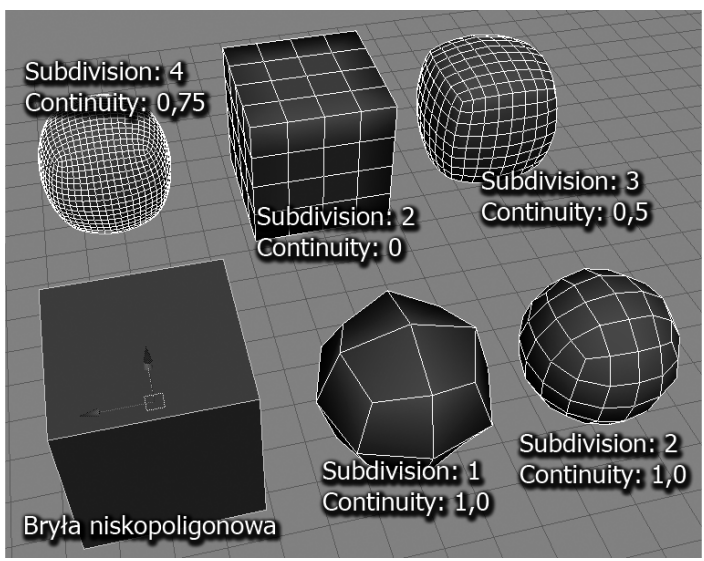
W różnych programach do grafiki 3D można spotkać się z różnymi algorytmami zagęszczania siatek, jednak większość z nich przy domyślnych ustawieniach działa w ten sposób, że równomiernie dzieli każdą ściankę źródłowego modelu na większą liczbę ścianek i ustawia je względem siebie w sposób zapewniający możliwie dużą gładkość powierzchni. Oznacza to jednak, że nie mamy bezpośredniej kontroli nad ostrością krawędzi i gęstością docelowej siatki w różnych jej miejscach. Innymi słowy, jeżeli poddamy zagęszczeniu nieskorygowane siatki niskopoligonowe z rysunku 2.11, to otrzymamy obłe bryły, w których program sprowadza wszystkie krawędzie modeli do możliwie okrągłych kształtów.



Rysunek 2.11. Siatki uproszczone i po zagęszczeniu do postaci wysokopoligonowej



Rysunek 2.12. Efekt automatycznego zagęszczenia siatek niskopoligonowych narzędziem Mesh/Smooth



Rysunek 2.13. Automatyczne wygładzanie sześcianu z różnymi ustawieniami narzędzia Smooth

W pakiecie Maya najbardziej podstawowym narzędziem do zagęszczania siatek jest polecenie *Mesh/Smooth* wybierane z górnej listwy menu. Jego najważniejsze parametry to *Subdivision Levels*, czyli liczba iteracji polegających na każdorazowym dzieleniu ścianek oraz Continuity, czyli, mówiąc w uproszczeniu, gładkość bryły. Jak widać na przykładzie zagęszczania sześcianu na rysunku 2.13, w zależności od ustawień tych dwóch parametrów uzyskujemy albo bryłę bardziej zbliżoną do sfery, albo prawie wcale nie wygładzoną, natomiast nie jesteśmy w stanie uzyskać efektu pudełka o delikatnie zaokrąglonych krawędziach jak na wcześniejszym rysunku 2.11.

Przedstawione wyżej problemy z uzyskaniem odpowiedniej krzywizny zaokrągleń wynikają z faktu, że automatyczne zagęszczanie prowadzi do powstawania siatki o możliwie regularnej topologii, co oznacza, że im bardziej równomierne będą rozmiary ścianek, tym bardziej obła stanie się bryła docelowa. Aby kontrolować stopień zaokrąglania różnych fragmentów modelu, wstawić należy nowe segmenty krawędzi i ścianek w miejscach, w których chcemy zmienić krzywizny wygładzonej siatki. Im mniejsze będą te nowe ścianki, tym ostrzejsze łuki możemy uzyskać po wygładzeniu.

Wstawianie nowych segmentów siatki można realizować dowolnymi narzędziami do edycji ściankowej, należy jednak przestrzegać przy tym co najmniej kilku podstawowych zasad:

- Automatyczne zagęszczanie wytwarza równomierną topologię siatki tylko w przypadku ścianek czworokątnych, dlatego powinniśmy unikać w bryle źródłowej ścianek o innej liczbie wierzchołków (trójkątów, pięciokątów itd). Nie zawsze jest to możliwe, lecz trzeba ograniczać liczbę ścianek innych niż czworokąty do minimum.
- Należy przez cały czas zachowywać możliwie równomierną strukturę siatki (unikać zbyt rozciągniętych ścianek, zbyt ostrych kątów między powierzchniami), gdyż zarówno narzędzia edycyjne programów graficznych jak i sam algorytm wygładzania dużo lepiej działają w przypadku siatek o regularnej topologii.
- Nowe segmenty należy wstawiać do siatki w takiej kolejności i w taki sposób, aby jak najdłużej mieć możliwość elastycznej kontroli nad wszystkimi aspektami kształtu siatki. Oznacza to, że w pierwszej kolejności wstawiamy segmenty odpowiedzialne za duże i regularne odkształcenia, a na samym końcu drobne i nieregularne zmiany.
- Warto rozważyć podzielenie modelu na więcej brył w zależności od ich struktury, kształtu oraz pożądanej szczegółowości. Każdy z elementów możemy edytować i wygładzać niezależnie, co często jest prostsze niż próby kontrolowania obiektu jako całości w jednej siatce.
- W przypadku brył symetrycznych można modyfikować jedną połówkę modelu i potem odbić ją względem płaszczyzny symetrii.
- Gdy model zawiera wiele powtarzających się elementów, najlepiej skupić się na edycji tylko jednego z nich a potem skopiować bryłę wynikową i rozmieścić kopie w odpowiednich miejscach.

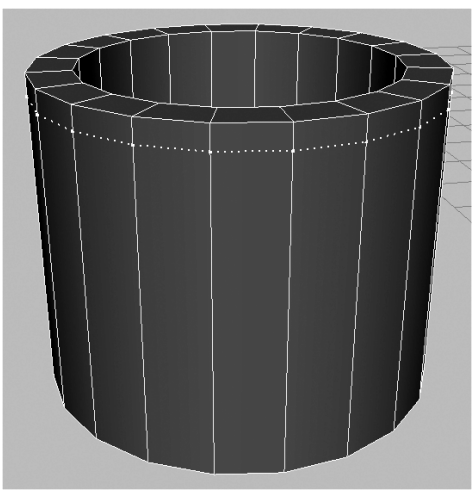
- Wstawianie i modelowanie nowych segmentów zazwyczaj prowadzi do powstania bryły zbyt skomplikowanej do bezpośredniego wykorzystania na potrzeby grafiki interaktywnej. Oznacza to, że zazwyczaj później trzeba osobno utworzyć zoptymalizowaną siatkę do wyświetlania w czasie rzeczywistym oraz osobno wersję wysokopoligonową, z której wygenerujemy normalmapy eksportowane do silnika gier. W praktyce pracujemy więc przeważnie z trzema różnymi modelami.
- Metody automatycznego wygładzania są bardzo wrażliwe na wszelkiego rodzaju błędy w siatkach (otwory, pokrywające się ścianki, ścianki o mikroskopijnej powierzchni itd). Przez cały czas pracy należy kontrolować siatkę i możliwie jak najwcześniej wykrywać wszelkie usterki.

Poniżej przedstawimy przykłady operacji umożliwiających uzyskanie brył wysokopoligonowych z rysunku 2.11. Efekty takie można uzyskać przy użyciu różnych kombinacji narzędzi do modelowania, tutaj zaś ograniczymy się do dwóch użytecznych funkcji, które dobrze sprawdzają się w tego typu przypadkach.

2.4 Wstawianie pętli krawędzi (Insert Edge Loop Tool)

Aby wymodelować rurę z zaokrąglonymi krawędziami wykonaj następujące czynności:

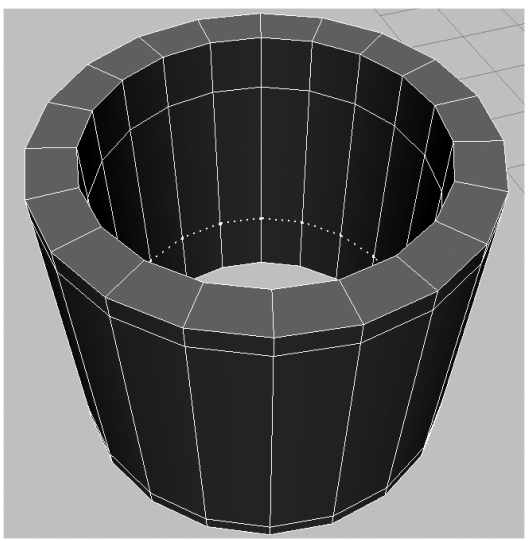
1. Utwórz w scenie podstawową bryłę o kształcie rury (*Create/Polygon Primitives/Pipe*).
2. Uaktywnij narzędzie do wstawiania pętli krawędzi (*Edit Mesh/Insert Edge Loop Tool*). Kliknij i przeciągnij kursorem na jednej z pionowych krawędzi



Rysunek 2.14. Wstawianie nowej pętli krawędzi

rury oraz wstaw nowy segment blisko zewnętrznej górnej krawędzi bryły. W trakcie przesuwania kursora program linią przerywaną będzie wizualizował sposób wstawienia nowej pętli (rysunek 2.14).

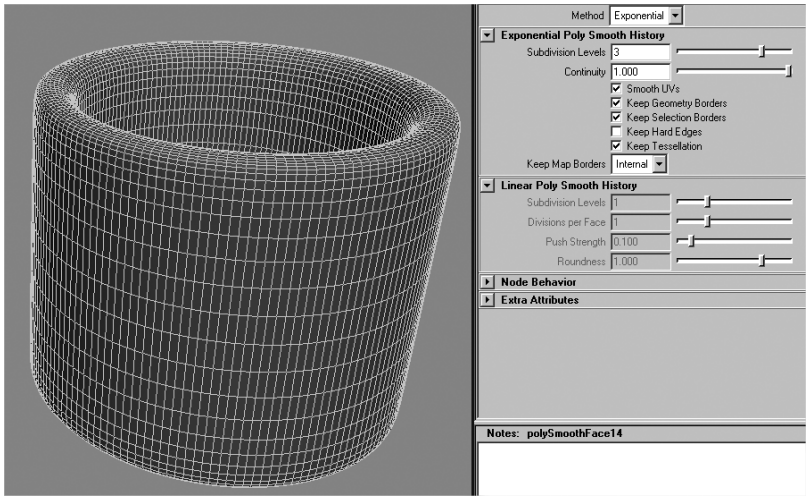
3. Powtórz tę operację jeszcze 3 razy, wstawiając dodatkowe pętle krawędzi u dołu po zewnętrznej stronie oraz u góry i u dołu po wewnętrznej stronie rury. Aby lepiej zobrazować sobie wpływ wykonywanych czynności na późniejszy efekt wygładzenia siatki, wstaw wewnętrzne segmenty w innej odległości od końców rury niż zewnętrzne (rysunek 2.15).



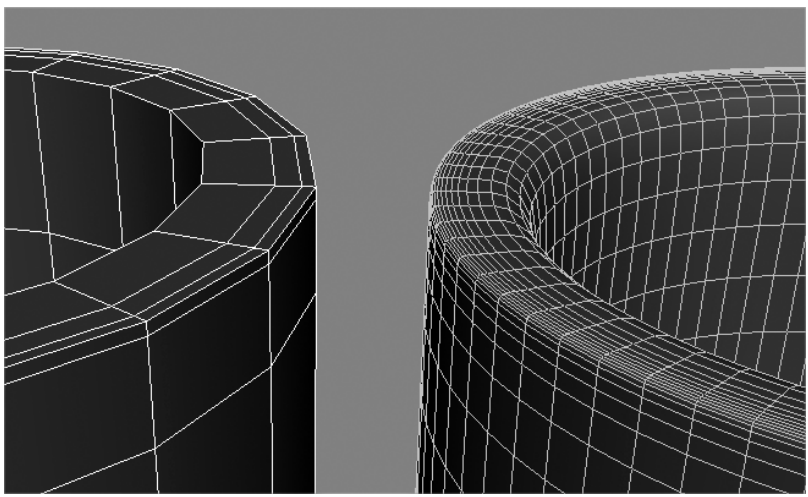
Rysunek 2.15. Wstawianie pozostałych pętli krawędzi

4. Z górnej listwy menu wybierz polecenie *Mesh/Smooth*. W zależności od liczby podziałów (*Subdivision Levels*) oraz gładkości (*Continuity*) uzyskasz efekt zbliżony do tego z rysunku 2.16.

O stopniu zaokrąglenia krawędzi przy końcach rury decyduje odległość pomiędzy dodatkowymi segmentami wstawionymi narzędziem *Insert Edge Loop Tool* a krańcowymi krawędziami bryły. Jeśli chcesz, aby brzegi powierzchni i na górze i na dole rury były jeszcze mniej obłe, cofnij operacje wygładzania i wstaw kolejny segment krawędzi (lub więcej segmentów) blisko brzegów siatki. Im więcej segmentów znajdzie się blisko załamania powierzchni i im bliżej będą one usytuowane względem siebie, tym ostrzejszy łuk powstanie po zaokrągleniu (rysunek 2.17). W praktyce zaś lepiej nie wstawiać zbyt wiele segmentów, tylko w miarę możliwości kontrolować promień zaokrągleń głównie poprzez odległości między segmentami.



Rysunek 2.16. Rura wygładzona narzędziem Smooth



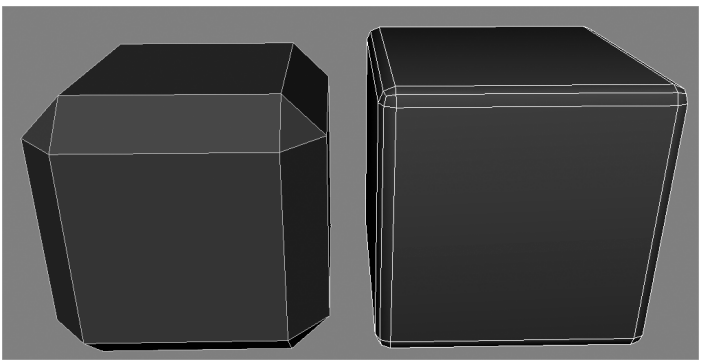
Rysunek 2.17. Aby uzyskać ostrzejszy profil na zewnętrznej krawędzi bryły, należy wstawić przy niej więcej ciasno ustawionych segmentów

2.5 Zaokrąglanie krawędzi (Bevel)

Innym sposobem wstawiania nowych segmentów i określania krzywizny wygładzonej bryły jest narzędzie do fazowania lub zaokrąglania krawędzi (*Bevel*).

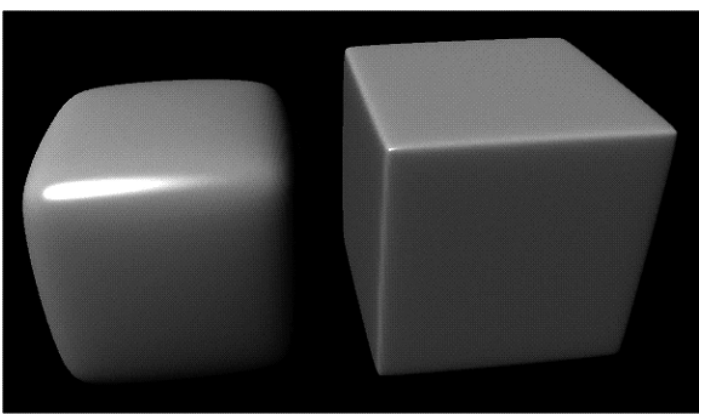
Aby kontrolować stopień zaokrąglenia krawędzi trójwymiarowego pudełka, wykonaj następujące czynności:

1. Utwórz prostopadłościan (*Create/Polygon Primitives/Cube*).
2. Wybierz polecenie Edit Mesh/Bevel i dobierz wartości parametrów *Offset* i *Segments* w taki sposób, aby uzyskać odpowiednio zaokrąglone krawędzie (rysunek 2.18).



Rysunek 2.18. Dwa sześciany z krawędziami zaokrąglonymi narzędziem Bevel przy różnych ustawieniach parametrów Segments i Offset

3. Z punktu widzenia poprawności topologii siatki dla późniejszych edycji lepiej jest użyć takich ustawień, przy których na rogach obiektu nie pojawią się trójkątne ścianki (jak na rysunku 2.18 po lewej).
4. Po dobraniu parametrów zaokrąglenia krawędzi, wybierz polecenie *Mesh/Smooth* i obejrzyj efekt wygładzenia (rysunek 2.19).

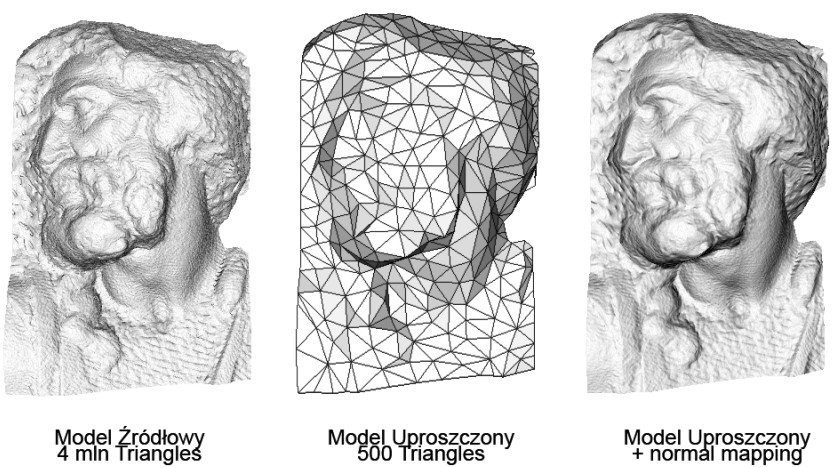


Rysunek 2.19. Efekt wygładzenia dwóch sześcianów o różnym promieniu zaokrąglenia krawędzi

3

Mapy normalnych

W poprzednich rozdziałach zajmowaliśmy się zagadnieniami związanymi z modelowaniem siatek niskopoligonowych a także skomplikowanych brył wysokopoligonowych. Obie te techniki modelowania są wykorzystywane we współczesnych grach i wizualizacjach architektonicznych. Obie też zazębiają się i przenikają w nowoczesnej grafice interaktywnej głównie za pośrednictwem obiektów tworzonych i wyświetlanych z wykorzystaniem normal mappingu (map normalnych, zwanych też z angielska normal-mapami).



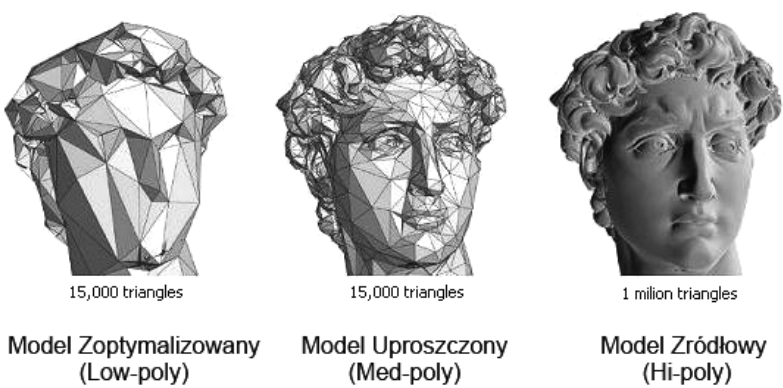
Rysunek 3.1. Porównanie złożonego obiektu wysokopoligonowego, uproszczonego modelu niskopoligonowego i wersji niskopoligonowej z mapą normalnych wygenerowaną w oparciu o złożoną siatkę

Mapowanie normalnych (ang. *normal mapping* lub *Dot3 bump mapping)* jest to technika teksturowania, która symuluje wypukłości powierzchni bez ingerencji w geometrię obiektu trójwymiarowego. Technika jest bardzo bliska tech-

nikom mapowania nierówności (ang. *bump mapping*) czy mapowania przemieszczeń (ang. *displacement mapping*), jednak pozwala osiągnąć znacznie lepsze wyniki przy stosunkowo małym użyciu pamięci i mocy obliczeniowej. Mapowanie normalnych polega na zastępowaniu oryginalnych wektorów normalnych ścianek (prostopadłych do powierzchni) wektorami normalnymi zapisanymi w odpowiedniej teksturze. Mapy normalnych, w przeciwieństwie do np. map nierówności (zapisywanych w skali szarości lub czarno-białych), są zapisane w składowych RGB. Każda składowa koloru mapy RGB, odpowiada współrzędnym składowej wektora normalnego danego piksela. Technika wykorzystywana w większości najnowszych gier i animacji.

3.1 Historia

Idea pobierania detali z wysokopoligonowych modeli została przedstawiona w „Fitting Smooth Surfaces to Dense Polygon Meshes" przez Krishnamurthy and Levoy, podczas targów SIGGRAPH w 1996 r. - podczas pokazu użyto map displacement narzuconych na obiekty typu NURBS. W roku 1998 dwie kolejne publikacje poruszyły temat transferu szczegółów z obiektów wysokopoligonowych do niskopoligonowych z użyciem normal map: „Appearance Preserving Simplification", Cohen oraz „A general method for preserving attribute values on simplified meshes", Cignoni. We wspomnianych opracowaniach naukowych zaprezentowany został również algorytm, który umożliwiał wykorzystanie techniki map normalnych w silnikach 3D służących do wyświetlania grafiki czasu rzeczywistego.



Rysunek 3.2. Podgląd wersji modelu o zróżnicowanej szczegółowości

Od strony technicznej sama idea normal mappingu nie uległa już dużym zmianom i jest w ciągłym użyciu w niewiele zmienionej formie do dnia dzisiejszego. Wraz z rozwojem i popularyzacją normal-mappingu zaczęły pojawiać się nowe

narzędzia dedykowane do ich obróbki - filtry do programów 2D, zewnętrzne generatory do programów 3D oraz wielofunkcyjne kompleksowe narzędzia jak xNormal. Wiodącą rolę w upowszechnieniu tej innowacyjnej techniki wyświetlania miały gry komputerowe i sprzętowe akceleratory graficzne, które od roku 1999 dawały już pełną możliwość budowania silników 3D wyświetlających grafikę z wykorzystaniem map normalnych.

Pierwszą platformą sprzętową (poza komputerami PC) wykorzystującą normal mapping była konsola Dreamcast koncernu SEGA. Następna generacja konsol - Microsoft Xbox oraz Sony PlayStation 2 z umiarkowanym sukcesem wykorzystywała dobrodziejstwa technologiczne normal mappingu. Prawdziwa rewolucja nastąpiła wraz z premierą konsol Xbox 360 i PlayStation 3, gdzie wcześniej wspomniana technologia w pełni rozwinęła swoje skrzydła i stała się standardem w produkcji grafiki do gier. W chwili obecnej zalety normal mappingu zaczynają wykorzystywać urządzenia przenośne, które stają się najprężniej rozwijającą się gałęzią przemysłu multimedialnego.

3.2 Jak działa normal mapping

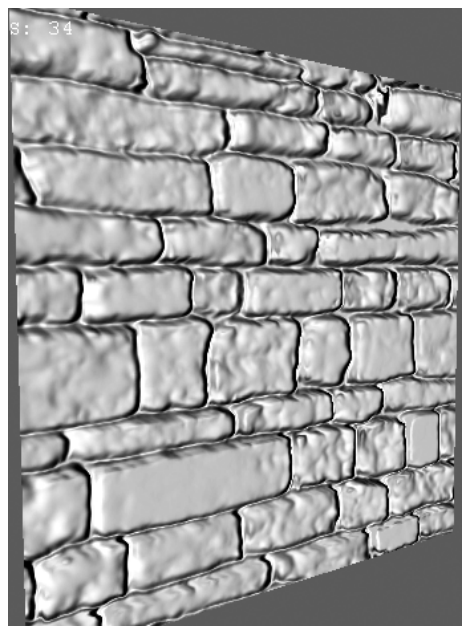
Idea działania normal mappingu jest stosunkowo prosta. Nie modyfikujemy geometrii prezentowanego obiektu, modyfikujemy zaś wektory normalne (czyli wektory prostopadłe do powierzchni) i zamiast interpolować wektory normalne dla każdego piksela (korzystając z wektorów normalnych w wierzchołkach modelu), odczytujemy je z dodatkowej tekstury. W jaki sposób? Każda składowa koloru piksela tekstury (RGB) odpowiada współrzędnym wektora normalnego (X,Y,Z) w renderowanym pikselu. Tak uzyskany wektor normalny jest używany w obliczeniach intensywności światła.

Oczywiście musimy pamiętać, iż normal mapping daje nam tylko złudzenie głębi, gdy spojrzymy na obiekt pod ostrym kątem to dostrzeżemy, że mamy do czynienia z płaską powierzchnią. Wrażenie „wypukłości" działa najlepiej gdy na obiekt patrzymy pod kątem prostym. Aby uniknąć tego wrażenia rozwinięto technikę wyświetlania poszerzając ja o parallax mapping (znany też jako offset mapping lub virtual displacement mapping) - pogłębia on wrażenie wypukłości tekstury, upodobniając ją do wypukłego reliefu (rysunki 3.3, 3.4).

3.3 Etapy tworzenia map normalnych

Pracę nad tworzeniem map normalnych z modeli wysokopoligonowych możemy podzielić na kilka etapów - po ich pozytywnym zakończeniu otrzymujemy gotową teksturę wykorzystującą techniczne dobrodziejstwa normal mappingu. Dodatkowo możemy jeszcze wzbogacić teksturę o efekt parallax mappingu, wypukłego reliefu, odblasków itp. celem polepszenia i pogłębienia efektu działania mapy normalnych.

Rysunek 3.3. Wyświetlanie tekstury wykorzystującej normal mapping



Rysunek 3.4. Wyświetlanie tekstury wykorzystującej normal mapping + parallax mapping

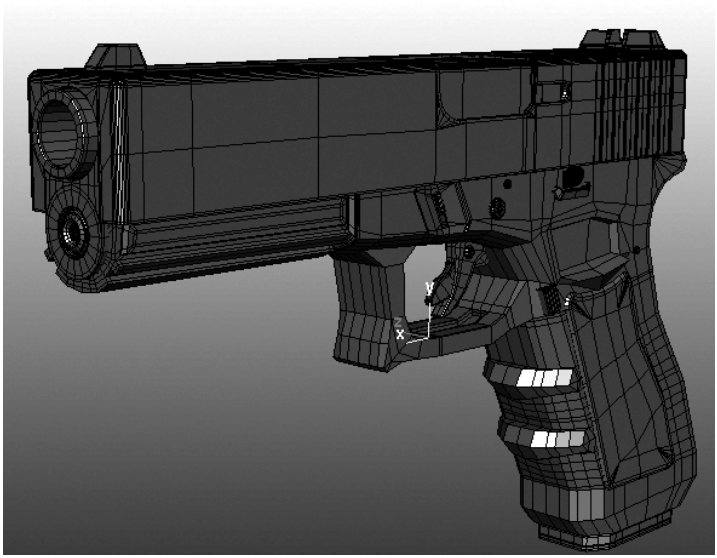
Etapy pracy

- **model wysokopoligonowy** (mid i hi-poly)
- **model niskopoligonowy** (low-poly)
- **mapowanie UV** (UV mapping)
- **wygładzanie** (Soft/Hard Edge)
- **wypalanie** (baking)
- **poprawki**

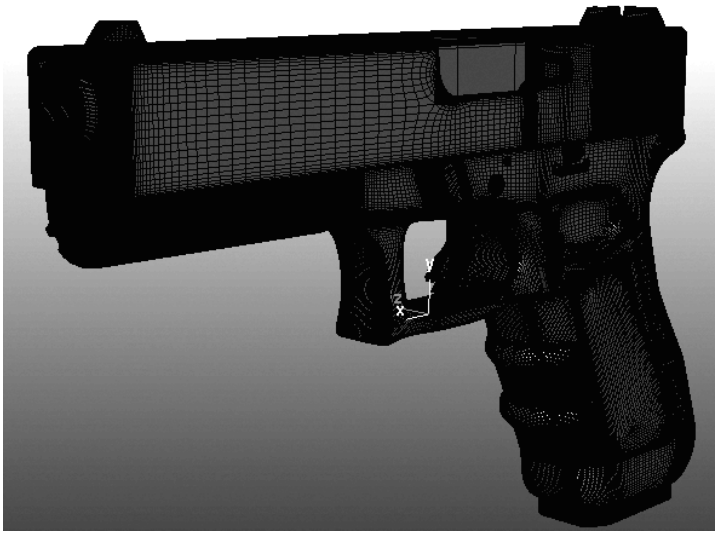
Aby operacja tworzenia mapy normalnych zakończyła się sukcesem, musimy wykazać się zegarmistrzowską precyzją wykonując wszystkie etapy pracy. Jeśli gdzieś popełnimy błąd lub nie dopatrzymy się uchybienia w modelu nasza praca bardzo się wydłuży, ponieważ często nie jest możliwe wprowadzenie poprawek na dowolnym etapie i konieczne staje się powtarzanie całych zadań od początku. Sama praca nad modelowaniem i tak jest już bardzo czasochłonna, dlatego proszę pamiętać o skrupulatności.

3.4 Modelowanie

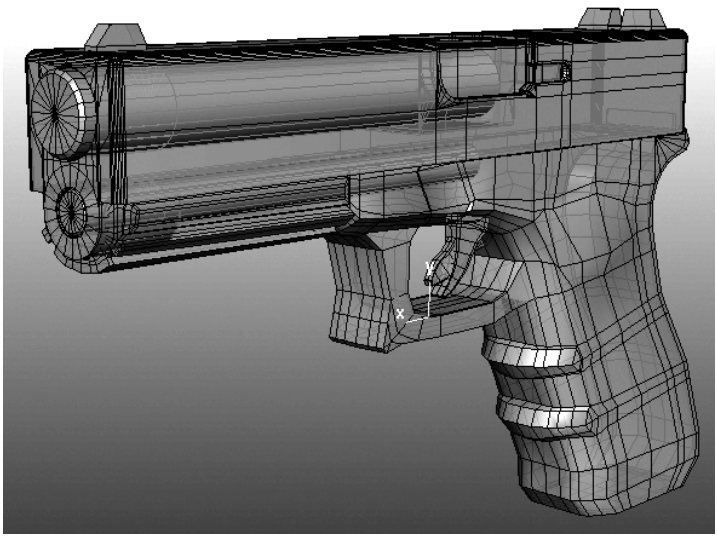
Musimy pamiętać, iż podczas prac nad modelem wykorzystującym mapy normalnych często trzeba korzystać z 3 różnych siatek. Pierwsza to siatka modelu przygotowana do zagęszczenia, druga to siatka wysokopoligonowa, z której pobierane są informacje o szczegółach geometrii, a trzecia siatka niskopoligonowa, na której wyświetlana jest mapa normalnych.



Rysunek 3.5. Siatka przygotowania do zagęszczenia (mid-poly) - ok. 30 tys. ścianek



Rysunek 3.6. Siatka wysokopoligonowa (hi-poly) - ok. 1,1 mln ścianek



Rysunek 3.7. Siatka niskopoligonowa (low-poly) - 6700 ścianek

Na rysunkach 3.5-3.7 widzimy porównanie 3 różnych siatek tego samego modelu.

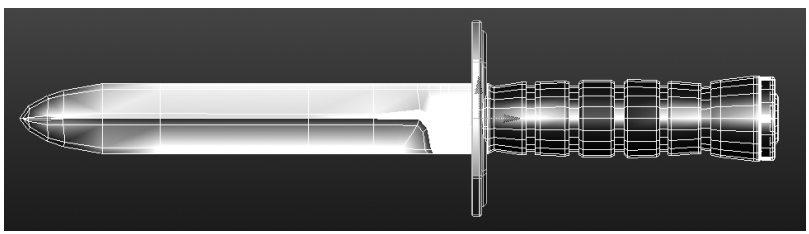
Na rysunku 3.7 widać model siatki zagęszczonej narzędziem Smooth, czyli wysokopoligonowej, której ilość trójkątów znacznie się zwiększyła - z blisko 30 tys. do ponad miliona.

Ostatnim modelem potrzebnym do naszej pracy jest model niskopoligonowy, czyli mocno zoptymalizowana siatka, który posłuży do rozłożenia współrzędnych UV i wygenerowania (wypalenia) mapy normalnych.

Na podstawie przykładu z nieco prostszym modelem prześledzimy krok po kroku etapy tworzenia mapy normalnych. Naszym obiektem testowym będzie model noża bojowego z kaburą.

Wybrane metody pracy nad modelami nisko i wysokopoligonowymi zostały omówione w poprzednich rozdziałach, poniższe przykłady ilustrują tylko poszczególne etapy pracy nad modelami na potrzeby normal mappingu.

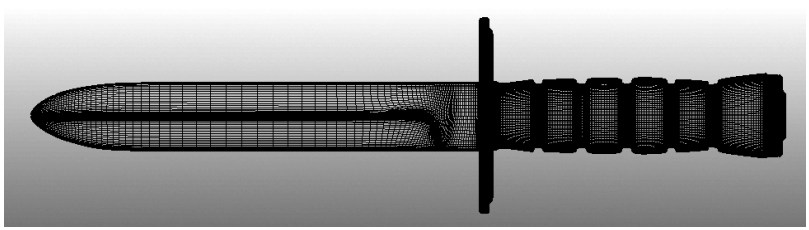
W pierwszej kolejności na podstawie odpowiednich referencji musimy wykonać model mid-poly tego noża - czyli siatkę która będzie odpowiednio przygotowana do automatycznego zagęszczenia (na przykład narzędziem Smooth, zobacz wcześniejszy rozdział).



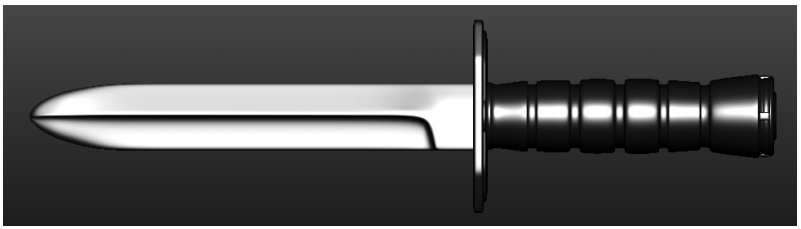
Rysunek 3.8. Siatka mid-poly przygotowana do zagęszczenia (3700 ścianek)

Kolejnym etapem jest zagęszczenie siatki czyli przygotowanie modelu źródłowego do wypalania map normalnych (rysunki 3.9-3.10).

Następny etap pracy, który musimy wykonać, to stworzenie modelu niskopoligonowego, na którym będziemy wyświetlać mapę normalnych pobranych z modelu zagęszczonego.

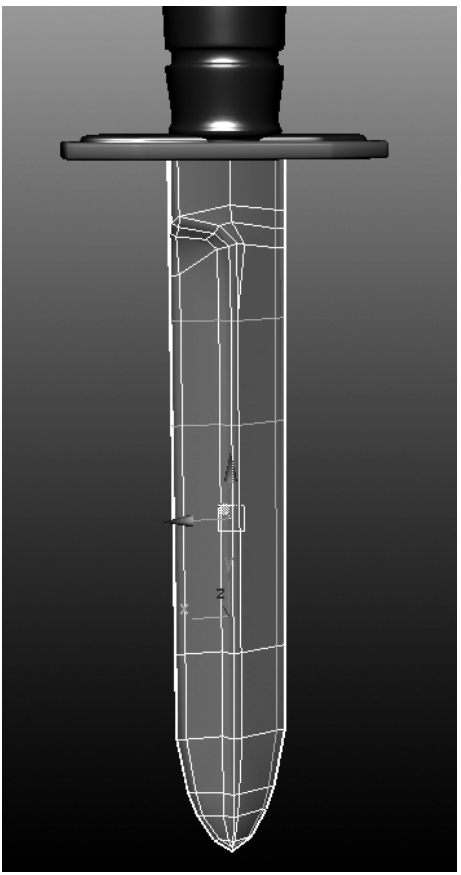


Rysunek 3.9. Zagęszczona siatka wysokopoligonowa (250 tys. ścianek)



Rysunek 3.10. Zagęszczona siatka wysokopoligonowa, podgląd modelu z materiałem

Siatkę modelu niskopoligonowego możemy wykonać na dwa sposoby: pierwsza polega na optymalizacji źródłowej siatki, druga zaś skupia się na stworzeniu nowego modelu z brył podstawowych.



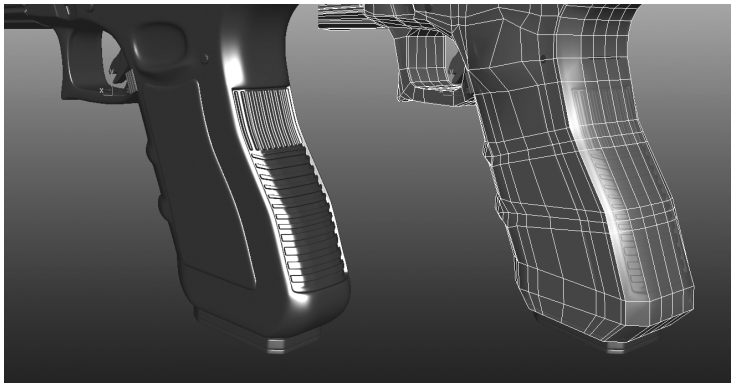
Rysunek 3.11. Edycja krawędzi - proces upraszczania siatki

W przypadku metody opartej na optymalizacji skupiamy się upraszczaniu oryginalnej siatki - czyli na usuwaniu i scalaniu wierzchołków oraz krawędzi modelu (na przykład funkcją *Merge* z górnego menu *Edit Mesh*). W ten sposób model noża staje się uproszczoną wersją źródłowego modelu i ma dużo mniej ścianek niż model zagęszczony, dzięki czemu można animować go w czasie rzeczywistym (rysunek 3.11).

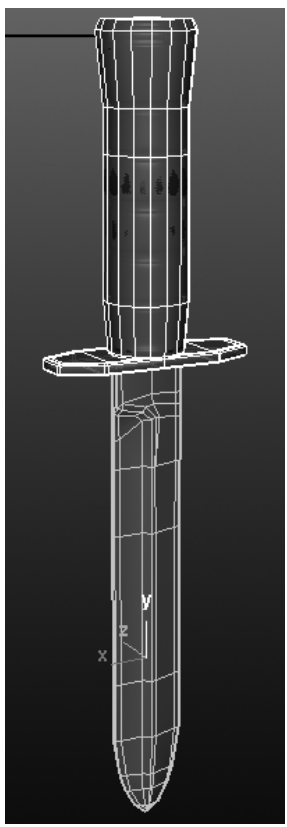
Do stworzenia modelu uproszczonego możemy również użyć zupełnie nowych siatek. Naszym celem jest przecież stworzenie nowego modelu który ma mniejszą liczbę ścianek - czy zrobimy to optymalizując źródłowy model, czy tworząc od podstaw nowy, nie ma większego znaczenia. Poniżej przedstawiono przykład gdzie model low-poly jest tworzony częściowo przez optymalizację (np. ostrze) jak i przez tworzenie nowych brył (uchwyt).

Tym sposobem ostatecznie uzyskaliśmy dwa modele: **niskopoligonowy** oraz **wysokopoligonowy**. Są one niezbędne do wygenerowania map normalnych.

Rysunek 3.12. Tworzenie modelu niskopoligonowego rozpoczęte w oparciu o nowe bryły



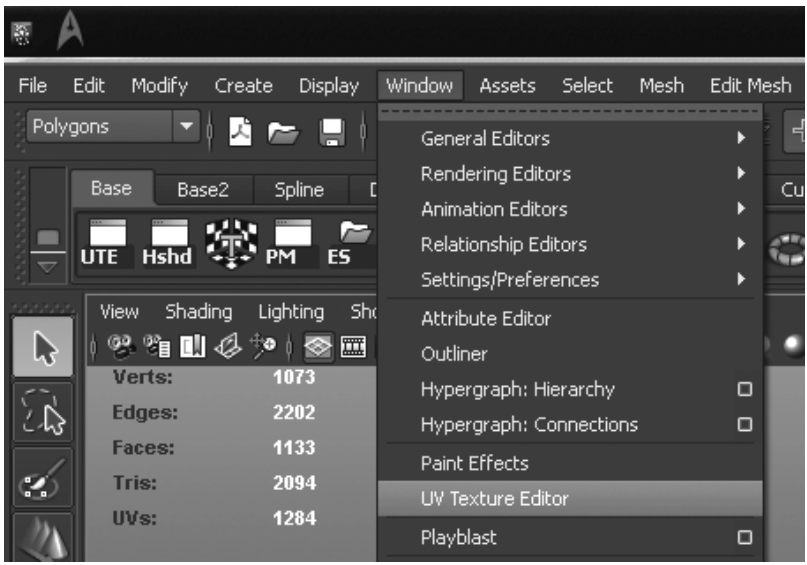
Rysunek 3.13. Kolejny przykład - porównanie siatek modelu wysokopoligonowego (po lewej) i niskopoligonowego (po prawej)



Rysunek 3.14. Finalna siatka modelu niskopoligonowego

3.5 Mapowanie UV

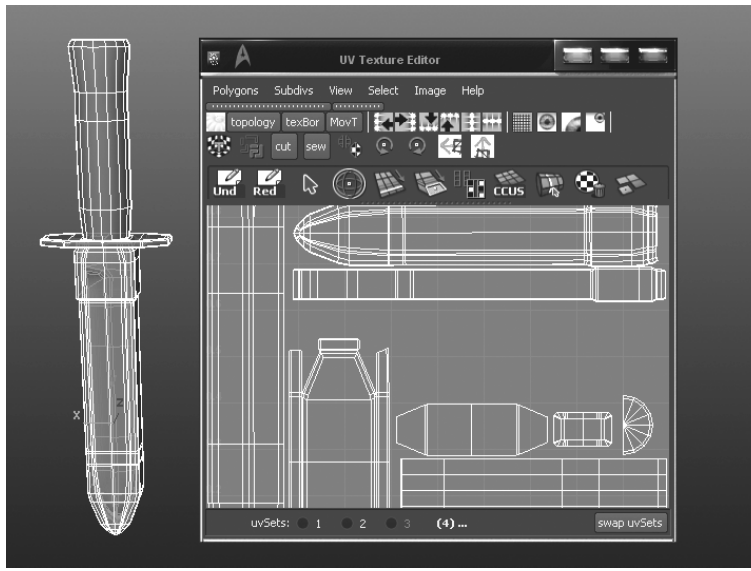
Gdy oba modele są już gotowe, możemy przejść do etapu mapowania - by wypalić teksturę normal-mapy musimy mieć dobrze przygotowane współrzędne mapowania UV modelu niskopoligonowego. W pracy nad mapowaniem w programie Maya pomaga nam *UV Texture Editor*, który jest osobnym edytorem wyspecjalizowanym w edycji mapowania. Dokładne omówienie zasad działania edytora wykracza poza ramy niniejszej książki. Żeby uruchomić edytor, musimy z górnego menu *Window* wybrać polecenie *UV Texture Editor* (rysunek 3.15). Na ekranie pojawi się wtedy okno edytora, w którym korzystając z dedykowanych narzędzi zmapujemy nasz model.



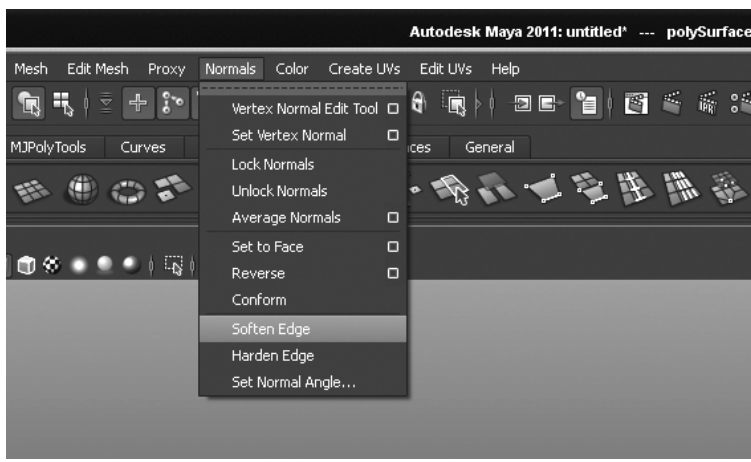
Rysunek 3.15. Otwieranie edytora współrzędnych UV

3.6 Wygładzanie

Ostatnim zadaniem, które musimy wykonać przed wypalaniem map normalnych, jest określenie odpowiedniego wygładzania naszego modelu low-poly. Zakres **wygładzania** krawędzi ustawiamy korzystając z narzędzi *Soften Egde, Harden Edge* lub *Set Normal Angle* w górnym menu *Normals*. Żeby wykonać operacje wygładzania, musimy najpierw zaznaczyć obiekt, a następnie z górnego menu wybrać jedną z tych funkcji, na przykład *Normals/Set Normal Angle* (rysunek 3.17).



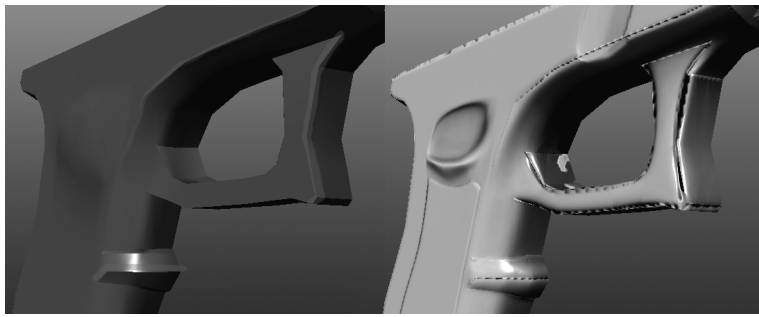
Rysunek 3.16. Okno UV Texture Editor wyświetlające zmapowaną siatkę modelu niskopoligonowego



Rysunek 3.17. Menu narzędzi wygładzania

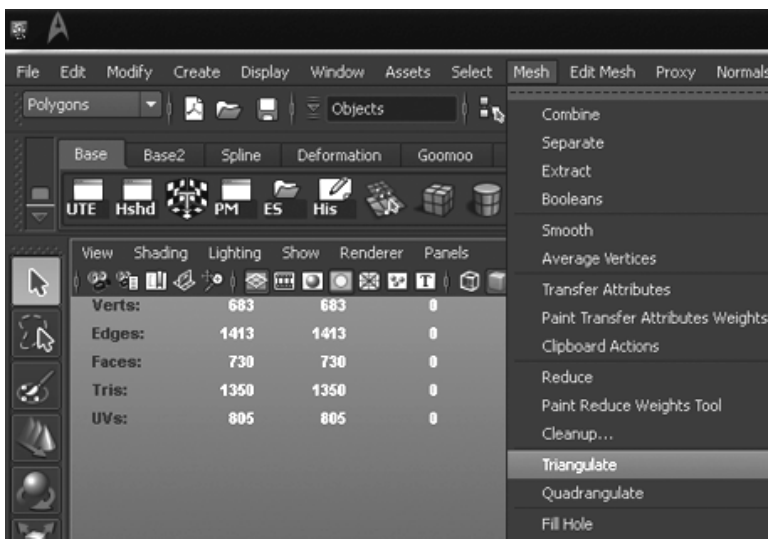
Musimy być świadomi, iż pozostawienie ostrych krawędzi na modelu low-poly może powodować błędy oraz przekłamania przy wypalaniu map normalnych. Najbezpieczniejszym wyjściem jest ustawienie kąta wygładzania na 180 stopni na całej siatce (czyli brak ostrych krawędzi), co wymaga wcześniejszego przystosowania topologii całej siatki do poprawnego wyświetlania przy takim

wygładzeniu. Tak wysokie wygładzanie sprawdza się najlepiej przy obiektach organicznych (czyli o obłych kształtach), ale najczęściej działa poprawnie także przy bardziej „technicznych" modelach i rozwiązuje wiele problemów przy samym wypalaniu normal-map.



Rysunek 3.18. Przykład ilustrujący błędy wypalania wynikające ze złego wygładzania siatki modelu - po lewej wyróżniono ostre krawędzie, których pozostawienie w modelu niskopoligonowym prowadzi do błędów w wyświetlaniu mapy normalnych (po prawej)

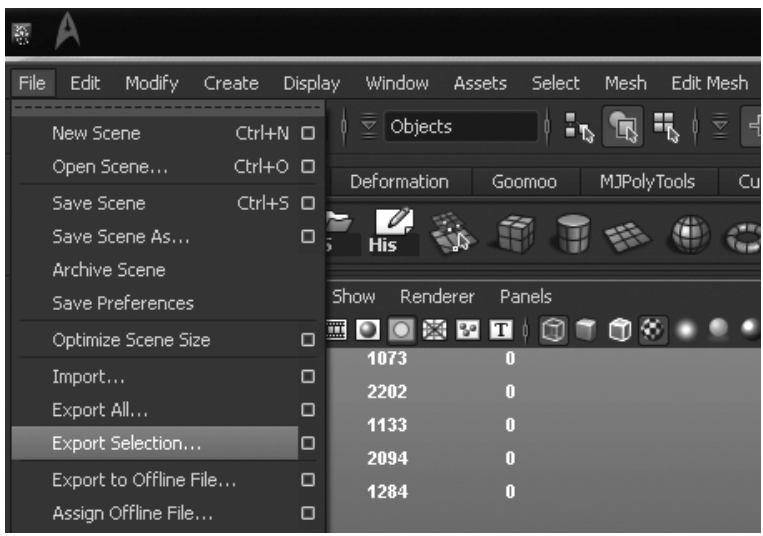
Wypalanie map normalnych można zrealizować w programie Maya, ale my posłużymy się zewnętrzną aplikacja. W tym przypadku, po zakończeniu pracy nad siatkami powinniśmy zmienić wszystkie ścianki w trójkąty (funkcja *Triangulate*) i wyeksportować model w formacie OBJ z programu Maya. Aby prze-



Rysunek 3.19. Wybór funkcji trójkątowania

prowadzić operacje trójkątowania należy zaznaczyć model i z górnego menu wybrać polecenie *Mesh/Triangulate* (rysunek 3.19). Operacja ta pozwala nam na unikniecie przekłamań w wyświetlaniu, pomaga nam w korektach topologi siatek przygotowywanych do operacji wypalania.

Ostatnim etapem jest przygotowanie modeli wysokopoligonowych i niskopoligonowych do eksportu. Model warto nieraz podzielić na kilka części a następnie wyeksportować je wszystkie w formacie OBJ - model noża podzielimy na 3 części (ostrze, uchwyt i pochwę) i wyeksportujemy te 3 elementy w wersji niskopoligonowej i wysokopoligonowej, czyli będziemy mieli w sumie 6 elementów (3+3). Aby wykonać operacje eksportu, należy zaznaczyć model, a następnie z górnego menu wybrać polecenie *File/Export Selection* (rysunek 3.20), z listy rozszerzeń *Files Type* wybierając format OBJ. Modele często eksportujemy w kilku odrębnych częściach w celu usprawnienia procesu wypalania map normalnych i uniknięcia błędów związanych z przenikaniem się poszczególnych siatek. Z tego powodu wypalanie map normalnych może być w praktyce wieloetapowe i w zależności od mapowania modelu, może być konieczne późniejsze złożenie efektów wypalania w pojedynczą teksturę w Photoshopie.



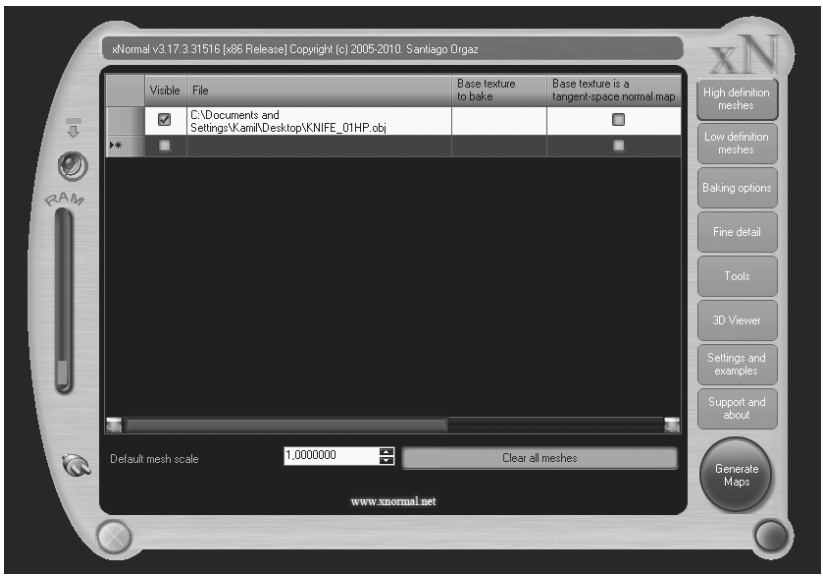
Rysunek 3.20. Wybór polecenia Export Selection

3.7 Wypalanie

Operacje wypalania normal-map możemy przeprowadzić z powodzeniem w pakiecie Maya, ale dużo większe możliwości dają nam darmowe pakiety do wypalania takie jak xNormal. Program ten ukierunkowany jest całkowicie na wypa-

lanie wszelkiego typu map, począwszy od normal-map poprzez tekstury ambient occlusion, height maps, cavity maps itd. Dodatkowo pakiet ten oferuje rozbudowane opcje usprawniające proces wypalania, których nie znajdziemy w pakietach Maya lub 3ds Max.

Proces wypalania jest bardzo prosty i intuicyjny, najpierw musimy uruchomić program xNormal po czym w oknie *High Definition Meshes* (rysunek 3.21)



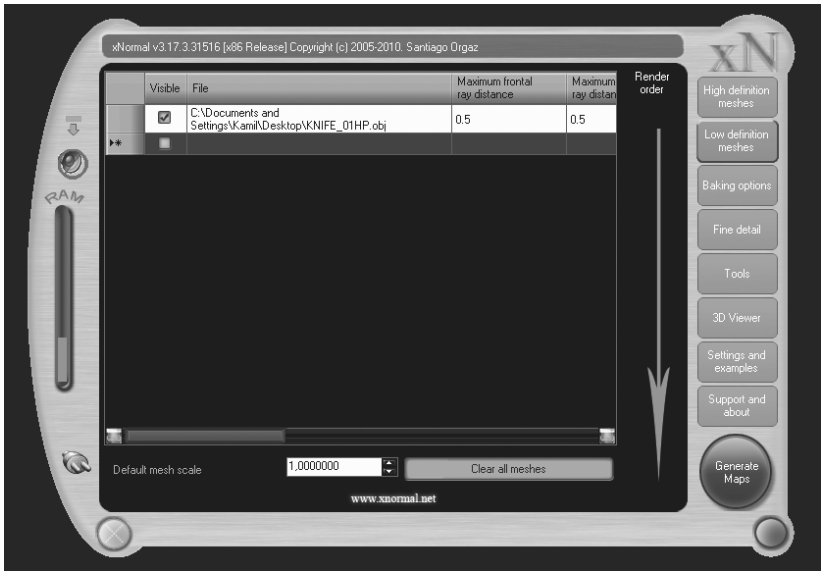
Rysunek 3.21. Okno High Definition Meshes programu xNormal

wskazać ścieżkę do OBJ modelu wysokopoligonowego. Następnie w oknie *Low Definition Meshes* (rysunek 3.22) podać ścieżkę do modelu niskopoligonowego.

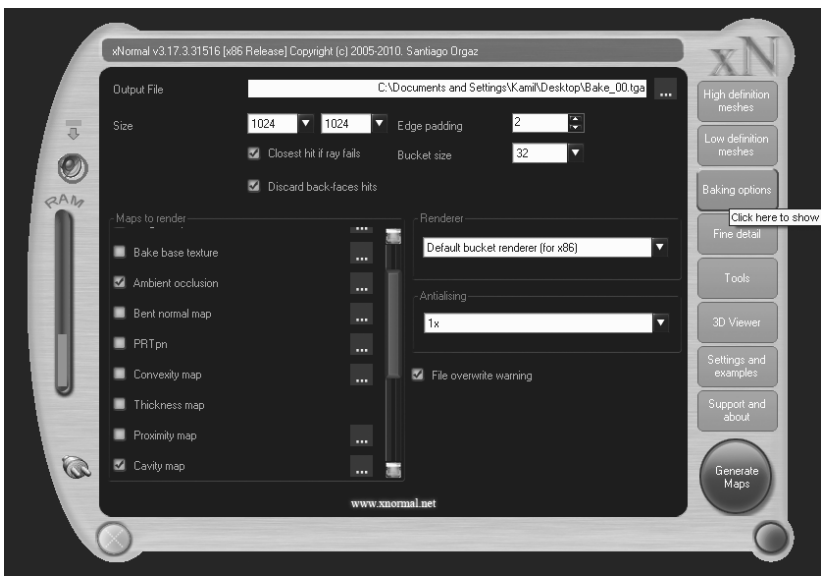
Kolejnym krokiem jest przejście do zakładki Baking Options - czyli opcji wypalania (rysunek 3.23). To tutaj wybieramy, jakie mapy z modelu wysokopoligonowego na model niskopoligonowy mają zostać wypalone (wygenerowane), określamy ich rozdzielczość, stopień kompresji itd.

Kiedy wszystko mamy już przygotowane, pozostaje nam zainicjować wypalanie, klikając na przycisku *Generate Maps*. Proces wypalanie zajmuje zazwyczaj kilka minut, lecz w zależności od ustawień i rozdzielczości docelowych tekstur może trwać nawet kilkanaście godzin (rysunek 3.24). Aby zaoszczędzić czas, zaleca się korzystanie z niższych rozdzielczości, szczególnie kiedy chcemy przetestować różne ustawienia *Baking Options* przed ostatecznym wygenerowaniem map w docelowej jakości.

Pozytywnie zakończony proces wypalania doprowadzi do utworzenia szeregu nowych map zrenderowanych z modelu wysokopoligonowego. Mapy te są po wyświetleniu na modelu niskopoligonowym stworzą wrażenie dużo większej

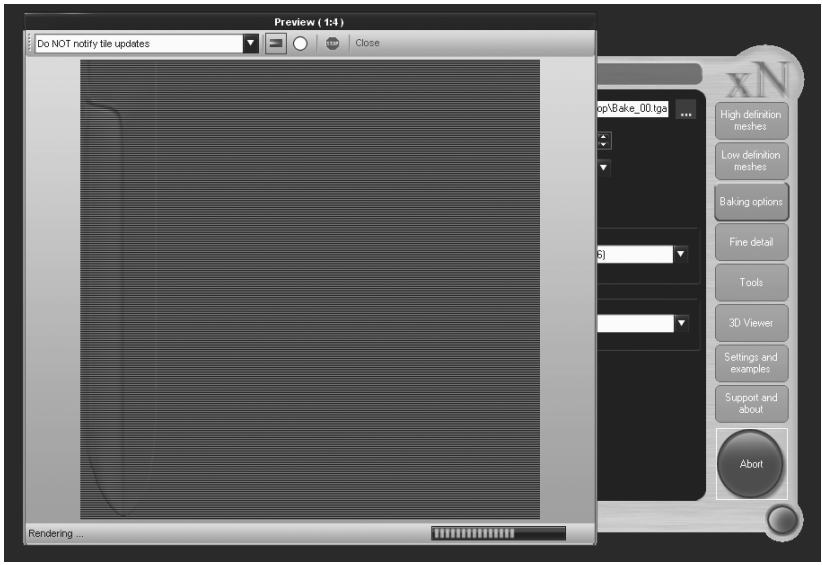


Rysunek 3.22. Okno Low Definition Meshes programu xNormal



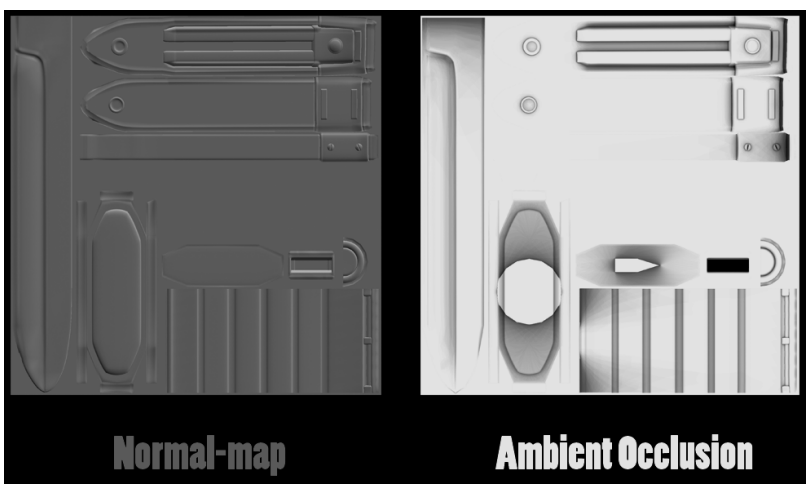
Rysunek 3.23. Okno wypalania - Baking Options

szczegółowości bry. W ten sposób dzięki wykorzystaniu techniki normal-mappingu wygląd obu modeli będzie zbliżony, pomimo faktu że model low-poly ma często kilkaset razy mniej ścianek niż model wysokopoligonowy.



Rysunek 3.24. Okno kontrolne podczas procesu wypalania

Najczęściej wypalamy mapy normalnych, jednakże często przydatne może być wypalenie przy okazji tekstur ambient occlusion i cavity, które znacznie usprawniają proces teksturowania (rysunek 3.25).



Rysunek 3.25. Podgląd wypalonych map Normal i Ambient Occlusion

Często zdarza się, że podczas procesu wypalania na mapach widoczne stają się drobne błędy i przekłamania, których nie opłaca się korygować poprzez czasochłonną edycję siatek w programie do modelowania. Wtedy najprostszym wyjściem jest skorzystanie z edytora do grafiki 2D, np. Photoshopa lub PaintShop-Pro i poprawienie niedoskonałości wygenerowanych wcześniej tekstur za pomocą narzędzi do malowania.

3.8 Wyświetlanie map normalnych

Jeśli wypalamy mapy normalnych w programie Maya, program automatycznie może dodać je do materiałów obiektu niskopoligonowego. Aby wyświetlić mapy normalnych wygenerowane w zewnętrznym pakiecie, takim jak xNormal, musimy stworzyć odpowiedni materiał własnoręcznie. Najpierw należy uruchomić edytor materiałów, *Hypershade*, a w nim korzystając z polecenia *Create/Materials/Phong* stworzyć nowy materiał (rysunek 3.26).

Rysunek 3.26. Tworzenie nowego materiału w edytorze Hypershade

Następnie musimy zaznaczyć model niskopoligonowy i przypisać mu utworzony wcześniej materiał (rysunek 3.27) - robimy to przytrzymując prawy klawisz myszy na ikonce materiału i wybierając polecenie *Assign Material To Selection*.

Rysunek 3.27. Przypisywanie materiału do modelu

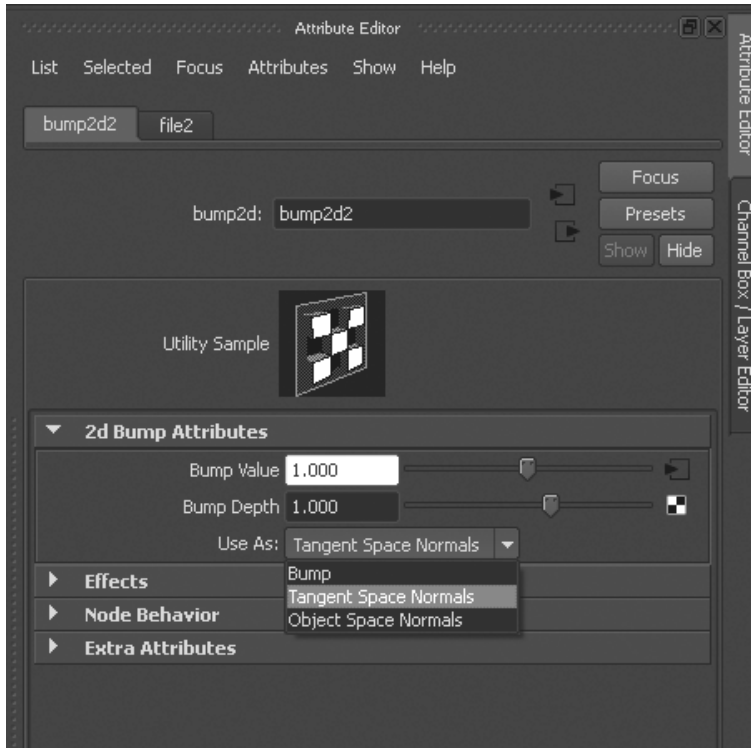
Teraz możemy już przejść do ustawienia w materiale stworzonych przez nas normal map, tekstur ambient itd. Aby to uczynić, musimy przejść do właściwości materiałów - w tym celu trzeba kliknąć dwukrotnie na kulce reprezentującej materiał, dzięki czemu otworzy się nowe okno *Attribute Editor* (rysunek 3.28). Tutaj klikając na szachownicy w polu *Color* wskażemy ścieżkę do tekstury koloru (może to być wstępnie mapa ambient occlusion), zaś klikają na zakładce *Bump Mapping* wskażemy ścieżkę do normal-mapy. Dodatkowo możemy tutaj jeszcze ustawić kolor, moc oraz skupienie połysku materiału - zakładka *Specular Shading*.

Aby mapa normalnych była poprawnie wyświetlana, musimy jeszcze włączyć opcję *Tangent Space Normals* (rysunek 3.29) we właściwościach *Bump Mapping* naszego materiału. Jeśli tego nie zrobimy, to nasza normal-mapa będzie wyświetlana jako standardowa mapa nierówności (*Bump*) co nie pozwoli uzyskać zamierzonego efektu.

Ostatnią rzeczą, którą musimy zrobić jest przestawienie trybu renderowania w oknie widokowym. By zobaczyć model niskopoligonowy z mapami normalnych, musimy z górnego menu nad oknem widokowym wybrać opcję *High Quality Rendering* (rysunek 3.30). W nowszych wersjach programu Maya mamy do dyspozycji opcję *Viewport 2.0*, która jeszcze dokładniej wyświetla materiały w czasie rzeczywistym. Na komputerach ze starymi kartami graficznymi, nieprawidłowymi sterownikami wyświetlania lub starszymi wersjami programu Maya, wspomniane powyżej opcje mogą nie działać lub też nie być dostępne w ogóle.

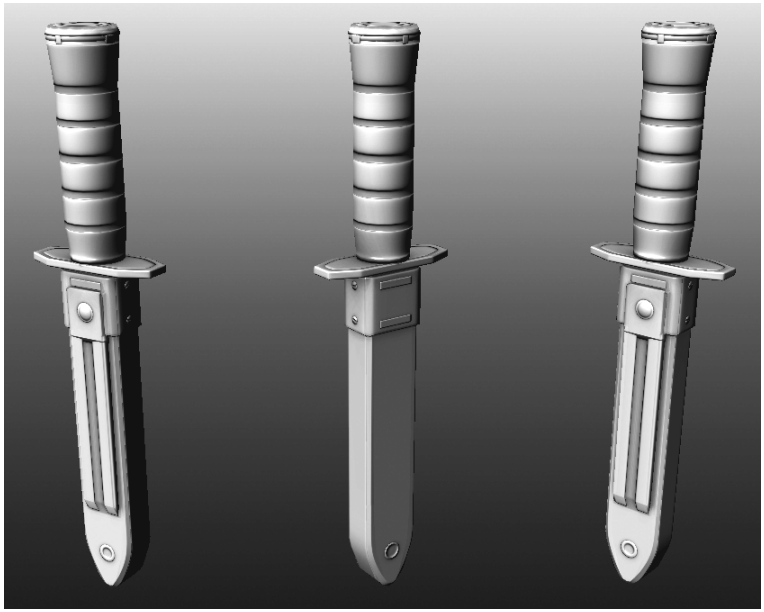
Rysunek 3.28. Edytor atrybutów (Attribute Editor) dla materiału Phong

Poniżej przedstawiamy kilka ujęć naszego niskopoligonowego z nałożoną na niego teksturą normal-mapy. Jak widać, z pewnej odległości wygląda on równie szczegółowo jak model wysokopoligonowy, pomimo że jest od niego kilkadziesiąt razy prostszy pod względem geometrii.

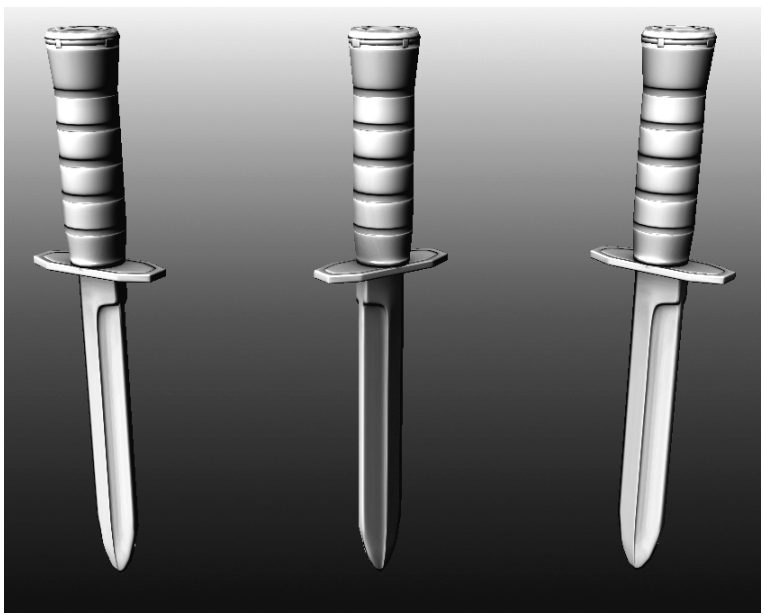


Rysunek 3.29. Opcje wyświetlania Bump Mapping

Rysunek 3.30. High Quality Rendering



Rysunek 3.31. Model niskopoligonowy z normal-mapą - wersja bagnetu z pochwą



Rysunek 3.32. Model niskopoligonowy z normal-mapą - wersja bagnetu bez pochwy

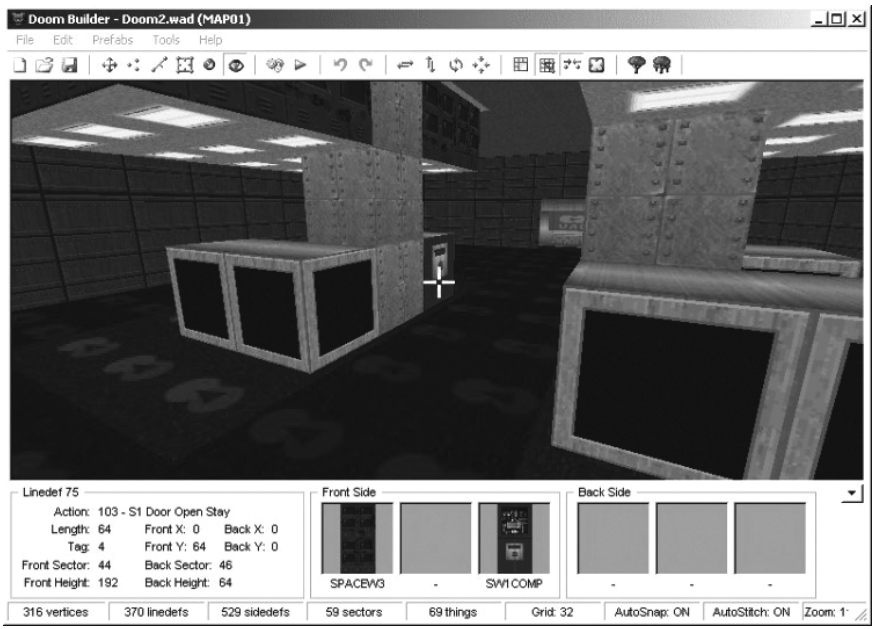
4

Modelowanie architektury w silnikach 3D

Od chwili kiedy pojawiły się na świecie pierwsze gry 3D, zaczęto również opracowywać narzędzia do ich produkcji oraz rozwijać silniki graficzne, które miały ułatwić i usprawnić proces ich tworzenia. Początkowo silniki, jak też narzędzia z nimi związane były tworzone tylko i wyłącznie do wewnętrznego użytku twórców gier i jako takie nie były udostępniane publicznie. Wraz z rozwojem świata multimedialnej rozrywki nastąpił też rozkwit wszelkiego typu edytorów do tworzenia elementów gier. W latach osiemdziesiątych dwudziestego wieku ekspansja komputerów 8-bitowych takich firm jak Atari i Commodore spowodowała rozwój tak zwanej „sceny modderskiej", czyli nurtu nieprofesjonalnych i niekomercyjnych gier tworzonych przez pasjonatów i hobbystów.

Rysunek 4.1. Obrazek z gry „Doom"

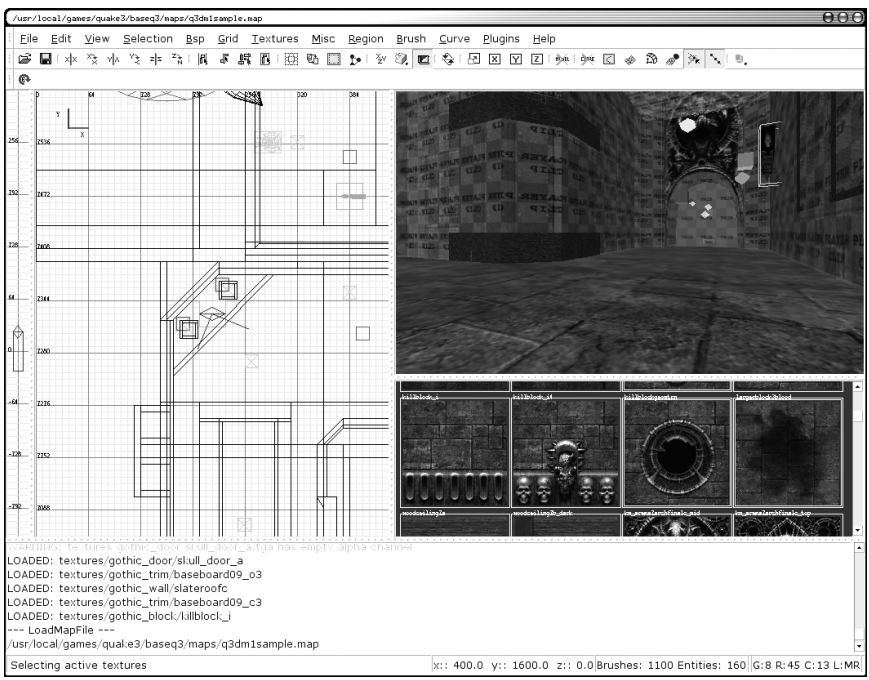
W latach dziewięćdziesiątych nastąpił gwałtowny rozwój gier 3D a wraz z nimi zaczęły się pojawiać pierwsze pół-profesjonalne edytory do gier dzięki, którym użytkownicy sami mogli dodać do swojej ulubionej gry nową architekturę, postać, teksturę albo muzykę. Edytory te, choć bardzo skomplikowane i mało przyjazne dla użytkowników, stanowiły krok milowy w rozwoju interaktywnej grafiki 3D i często stanowiły pierwsze pole kontaktu z tym zagadnieniem dla przyszłych projektantów i artystów. Jedną z pierwszych gier, która została wzbogacona o edytor 3D, był „Doom" firmy Id Sofware, wydany w roku 1993. Dzięki licznej rzeszy pasjonatów, społeczność graczy-twórców skupiona wokół gry mocno się rozrosła co zaowocowało dużą ilością darmowych dodatków takich jak zbiory postaci, nowych elementów architektury itd.



Rysunek 4.2. Doom Builder - edytor 3D gry „Doom"

W 1996 roku firma Id Software wydała grę akcji „Quake", która uznawana jest za pierwsza grę akcji zrealizowaną w pełnym 3D - czyli taką, gdzie wszystkie elementy wirtualnego świata (otoczenie, postacie, itd.) były bryłami przestrzennymi wprawionymi w ruch za pomocą silnika 3D wyświetlającego grafikę w czasie rzeczywistym. „Quake" w momencie premiery zadziwiał możliwościami technologicznymi silnika gry, który nie miał wówczas sobie równych, a to pobudzało umysły wielu pasjonatów, którzy chcieli tworzyć swoją architekturę, postacie i tekstury do popularnego QuakeWorld. W niedługim czasie zaczęły się pojawiać nieoficjalne edytory do tworzenia grafiki, jednym z bardziej

popularnych był 3D Match Maker, służący do budowania lokacji do rozgrywki wieloosobowej. W owym czasie spopularyzowały się również słowa „mapa" i „level", które oznaczały nowe lokacje i misje do gry - ludzi, którzy zajmowali się tworzeniem tego rodzaju grafiki zaczęto nazywać po angielsku level designerami lub map-makerami. W kolejnych latach, za sprawą takich firm jak Epic Games czy Valve Software następował gwałtowny rozwój kolejnych rozbudowanych silników i edytorów 3D.



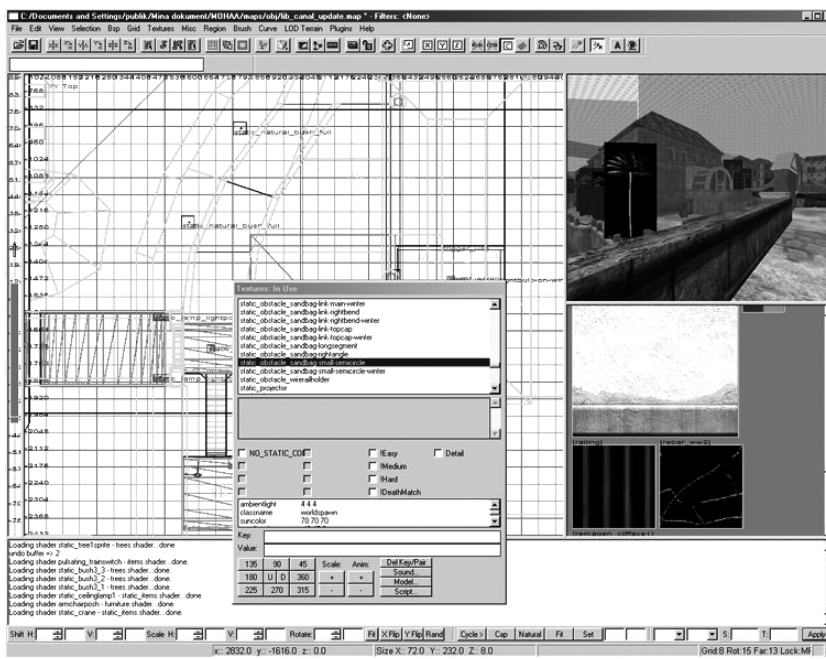
Rysunek 4.3. GTK Radiant - edytor 3D gry „Quake"

Jeśli chodzi o terminologię, to możemy powiedzieć, iż w grach komputerowych za przetwarzanie grafiki odpowiada tzw. silnik graficzny zwany potocznie silnikiem 3D, a do tworzenia grafiki, którą wyświetla silnik używa się wewnętrznych lub zewnętrznych edytorów 3D.

4.1 Silniki graficzne

Silniki można go podzielić na dwa podstawowe rodzaje: silniki 2D i silniki 3D. W niektórych przypadkach silnik 2D może prezentować grafikę bardziej atrakcyjną niż silnik 3D, jednak wraz ze wzrostem możliwości kart graficznych w dziedzinie przetwarzania grafiki silniki 2D wychodzą z użycia lub pozostają

Rysunek 4.4. Obrazek z silnika 3D gry „Quake"



Rysunek 4.5. GTK Radiant - edytor 3D gry „Medal of Honor"

domeną urządzeń przenośnych jak np. telefony komórkowe. Należy też nadmienić, iż aktualne możliwości techniczne sprawiają, że pisanie dedykowanych silników 2D do gier coraz rzadziej się opłaca, bowiem w wielu przypadkach dużo lepszym rozwiązaniem wydaje się zaangażowanie podsystemu grafiki 3D do wyświetlania grafiki dwuwymiarowej. W takim układzie moduł odpowiedzialny za tworzenie oraz edycję grafiki 2D staje się integralną częścią głównego silnika 3D.

Warto wspomnieć, że w obecnych czasach silniki 3D zaprojektowane pierwotnie do gier są wykorzystywane na szeroką skale w prezentacjach architektonicznych, rekonstrukcjach, aplikacjach szkoleniowych, internetowych i wielu innych dziedzinach, w których grafika interaktywna ma przewagę nad grafiką statyczną.

4.1.1 Silnik 2D

Tego typu silnik zazwyczaj wyświetla scenę w grze poprzez nakładanie na siebie gotowych dwuwymiarowych obrazów w odpowiedniej kolejności. Niekiedy są one dodatkowo modyfikowane w czasie rzeczywistym, np. poprzez zmianę jasności, maskowanie barw lub przeźroczystości. Elementy sceny mogą zostać stworzone przez twórców poprzez wyrenderowanie obiektów trójwymiarowych lub z dowolnej grafiki rastrowej lub wektorowej.

Silnik 2D przetwarza jedynie grafikę dwuwymiarową, jednak twórcom gier zazwyczaj zależy, by utworzona w ten sposób scena sprawiała wrażenie trójwymiarowej. Wiele silników 2D wykorzystuje rzut izometryczny. Świat przedsta-

Rysunek 4.6. Obrazek z silnika 2D gry „Another World"

wiony w ten sposób może prezentować się atrakcyjnie, często bardziej szczegółowo niż w przypadku grafiki 3D czasu rzeczywistego. Podstawową wadą tego typu wyświetlania grafiki jest brak perspektywy (w widoku izometrycznym te obiekty, które w świecie gry mają sprawiać wrażenie bardziej oddalonych, są tej samej wielkości co obiekty bliższe). Innym istotnym ograniczeniem z punktu widzenia aplikacji interaktywnych jest brak możliwości swobodnego obracania kamery w grze.

Silniki 2D są zazwyczaj stosunkowo proste w tworzeniu. Głównymi kwestiami podczas wyświetlania zawartości gry jest określenie miejsca poszczególnych elementów sceny na ekranie oraz kolejności ich wyświetlania. Łatwo jest również z góry określić, które elementy sceny nie są widoczne, bo np. znajdują się poza ekranem. Powoduje to, że optymalizacje silników 2D są mało skomplikowane. Dotychczas silniki 2D były bardzo popularne w grach strategicznych, jednak wraz ze wzrostem jakości grafiki 3D także na tym polu zaprzestaje się tworzenia gier 2D.

4.1.2 Silnik 3D

Jego zadaniem jest przygotowanie i wyrenderowanie sceny 3D. Grafika oparta na silniku 3D jest zazwyczaj atrakcyjna dla gracza, jednak stworzenie dobrego silnika 3D jest niezwykle trudnym procesem. Sam rendering obiektów jest zadaniem tak złożonym, że niemal wszystkie tworzone obecnie gry wykorzystują gotowe biblioteki programistyczne, jak DirectX czy OpenGL. Dzięki temu programista nie musi tworzyć kodu odpowiedzialnego za większość podstawowych funkcji związanych z renderingiem, nie musi też zagłębiać się w szczegóły działania kart graficznych.

Podstawowymi elementami, z których tworzony jest świat gry, są obiekty trójwymiarowe, przeważnie tworzone w programach takich jak Maya lub 3ds Max oraz tekstury nakładane na te obiekty. Niezależnie od tego we współczesnych grach występują także różne efekty dodatkowe, jak np. światła, odblaski cieniowanie czy też cząsteczki. Efekty te wpływają na realizm grafiki oraz na prędkość tworzenia poszczególnych klatek sceny. Z upływem lat w grach komputerowych pojawia się coraz więcej efektów wizualnych, co sprawia, że utworzenie silnika 3D staje się procesem coraz bardziej pracochłonnym. Tworzenie takich efektów jak kurz, falowanie wody, refleksy świetlne, pochłania wiele czasu ze strony grafików i programistów, wymaga również rozbudowywania układów graficznych o coraz bardziej zaawansowane technologie.

Ograniczona prędkość przetwarzania grafiki powoduje, że twórcy gier poświęcają mnóstwo czasu na optymalizację silników 3D. Jest to zagadnienie bardzo obszerne. Jednym z najistotniejszym elementów optymalizacji jest określenie z góry, które elementy świata gry nie będą widoczne. Trudno jest stwierdzić na przykład czy dom, na który patrzymy, na pewno zasłania w całości dom znajdujący się dalej. W przypadku gier, w których gracz porusza się po zamkniętych pomieszczeniach, określa się je jako odseparowane elementy. Miejsca,

przez które jeden pokój może być widziany z innego pokoju, tworzą tzw. portale widoczności. Dzięki temu, jeśli gracz znajduje się w którymś z pokoi, to można pominąć wyświetlanie innych pokoi, chyba że w zasięgu widoku znajduje się portal. Taki sposób optymalizacji ma ogromny wpływ na szybkość działania gry. Jednym z prostszych sposobów optymalizacji, dzięki któremu można zredukować ilość wyświetlanych elementów, jest mgła.

Rysunek 4.7. Obrazek z silnika 3D gry „Doom 3"

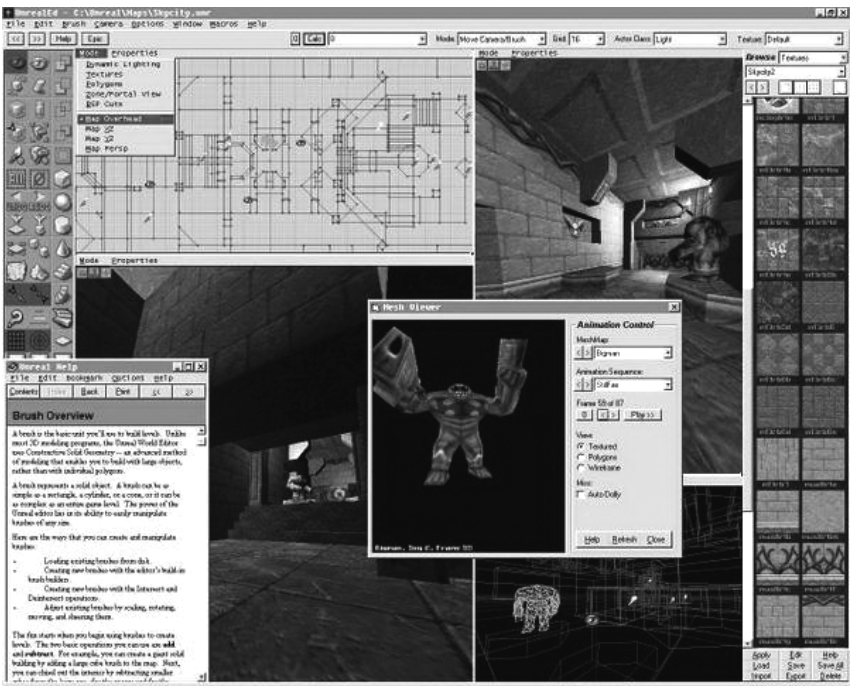
Kolejną metodą optymalizacji jest stosowanie tekstur o różnych wielkościach i nakładanie ich na obiekty w zależności od odległości od kamery. Czasami stosuje się również obiekty o różnej liczbie wielokątów w zależności od dystansu do gracza (ang. *level of detail*). O ile przeskalowanie tekstur jest proste, o tyle zmniejszenie ilości wielokątów w obiekcie 3D przy zachowaniu podobnego wyglądu siatki z nałożoną teksturą jest dość trudne. W silnikach 3D stosuje się jeszcze cały szereg innych optymalizacji, jak np. renderowanie niektórych elementów sceny rzadziej niż w każdej kolejnej klatce itd.

Należy tutaj wspomnieć, iż w skład kompleksowego silnika 3D zwanego potocznie silnikiem gry (z angielskiego „*game engine*") wchodzą też moduły odpowiedzialne za dźwięk, muzykę, sztuczną inteligencję, symulacje fizyczne, wykrywanie kolizji, itd. Silnik gry zajmuje się interakcją elementów gry. Jest on często błędnie utożsamiany z silnikiem graficznym lub silnikiem 3D a w rzeczy-

wistości jest to główny i najważniejszy moduł integrujący mniejsze podmoduły. Firmy deweloperskie często korzystają z gotowych silników, przy czym zawsze silnikowi towarzyszą narzędzia, dzięki którym można stworzyć pewne elementy gry bez ingerencji w kod źródłowy silnika.

4.2 Rys historyczny

Wraz z rozwojem komputerów oraz akceleratorów 3D takich, jak na przykład 3Dfx VooDoo, w połowie lat dziewięćdziesiątych dwudziestego wieku zaczęły pojawiać się gry wykorzystujące na szeroką skalę zalety technologii 3D. Wraz z sukcesami takich tytułów jak „Doom" lub „Duke Nukem 3D" spopularyzowały się też edytory 3D, dzięki którym można było tworzyć nowe wirtualne lokacje, wzbogacać architekturę i postacie o nowe tekstury i obiekty. Połowa lat 90. XX wieku to okres przełomowy tak w przemyśle gier jak i w tworzeniu grafiki 3D. Właśnie wtedy stało się możliwe tworzenie w pełni trójwymiarowych gier, bogatych i urozmaiconych tekstur, lokacji przepełnionych grą świateł oraz cieni przeliczanych w czasie rzeczywistym, postaci z animacją szkieletową itd. To właśnie od tamtego czasu światy gier zaczęły stawać się coraz bardziej sugestywne i kompleksowe, a ogromne rzesze ludzi zafascynowanych możliwościami



Rysunek 4.8. Obrazek z edytora 3D gry „Unreal"

tych nowych technologii zaczęło implementować do nich własną grafikę. W kolejnych latach zakres zastosowań silników 3D znacznie się rozszerzył, zaczęto ich używać do prezentacji architektonicznych, prezentacji multimedialnych, rekonstrukcji archeologicznych, interfejsów przeglądarkowych i wielu innych zastosowań.

Elementem łączącym artystę grafika z silnikiem gry jest edytor 3D - narzędzie to umożliwia projektantom tworzenie nowych lokacji lub integrację zewnętrznej grafiki (np. modeli stworzonych w Maya) ze środowiskiem gry, programowanie rozgrywki, dodawanie efektów specjalnych, animacji itd. To właśnie praca w edytorze zbiera wszystkie elementy składowe w całość tworząc grę, to tutaj z setek różnych elementów komponuje się grafikę w postaci jaką oglądamy w finalnej wersji gry, projektuje rozgrywkę oraz dodaje efekty oświetlania w czasie rzeczywistym itp. Edytor jest interfejsem między projektantami a silnikiem gry.

4.3 Tworzenie interaktywnej grafiki w edytorze 3D

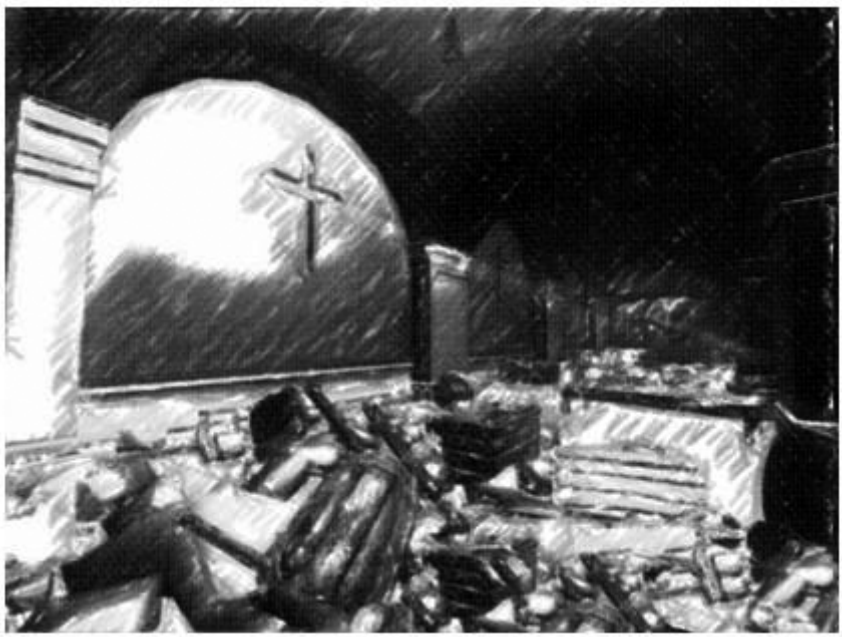
Ponieważ tworzenie architektury w edytorach 3D takich jak np. PainED, Radiant lub UnrealdED, różni się znacznie w zależności od wybranego narzędzia, w poniższym tekście skupimy się wyłącznie na bardziej ogólnych założeniach, które musimy spełnić aby stworzyć dobrze wyglądającą lokacje w silniku 3D.

Prace nad tworzeniem lokacji 3D możemy podzielić na kilka istotnych etapów, opisanych poniżej.

4.3.1 Faza koncepcyjna

Jednym z pierwszych etapów pracy nad tworzeniem lokacji do gry jest stworzenie szkicu lub szkiców koncepcyjnych. Oczywiście najpierw jest pomysł, wszyscy projektanci rozmawiają razem jaka lokacja do danej gry będzie najlepsza - należy pamiętać, iż otoczenie w grze musi także spełniać wymogi grywalnościowe (z angielskiego - *gameplayowe*), czyli otoczenie musi być tak zaprojektowane, by swobodnie mógł się po nim poruszać gracz oraz przeciwnicy i ruchome elementy świata gry. Załóżmy, że naszą lokacją będzie opuszczona forteca. Na etapie pracy koncepcyjnej tworzymy pierwsze szkice - tak rzuty techniczne jak np. z góry jak i kolorowe obrazki z ujęciami panoramicznymi. Dodatkowo też wspieramy się każdym możliwym źródłem referencji - komiksem, opisami z książek historyczny czy też fabularnych, zdjęciami, rysunkami historycznymi itd. W ostatecznej fazie koncepcyjnej powinniśmy mieć już jasną i czytelną wizje lokacji, którą chcemy stworzyć, popartą dokładnym planem, szkicami, referencjami i spójną z wizją grywalnościową oraz technicznymi wyznacznikami projektu.

Podsumowując - wizja artystyczna i techniczna muszą się z sobą zazębiać, aby razem stworzyć spójną całość.

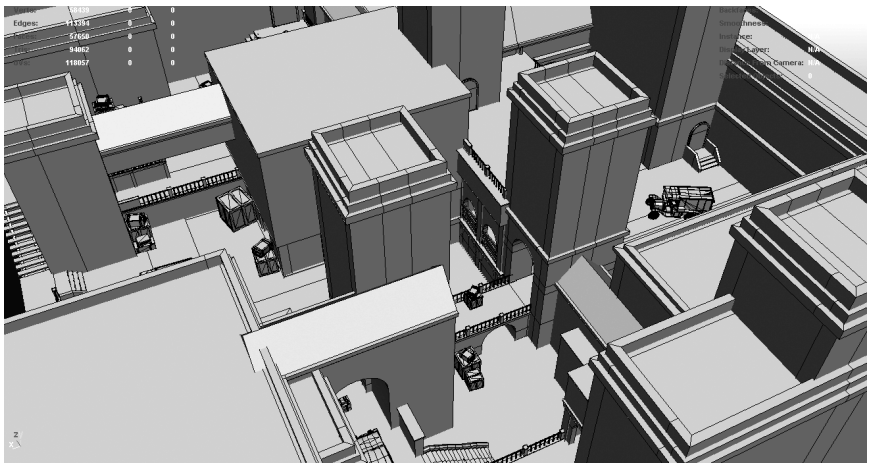


Rysunek 4.9. Szkic koncepcyjny z gry „NecroVisioN"

4.3.2 Szkic 3D (makieta)

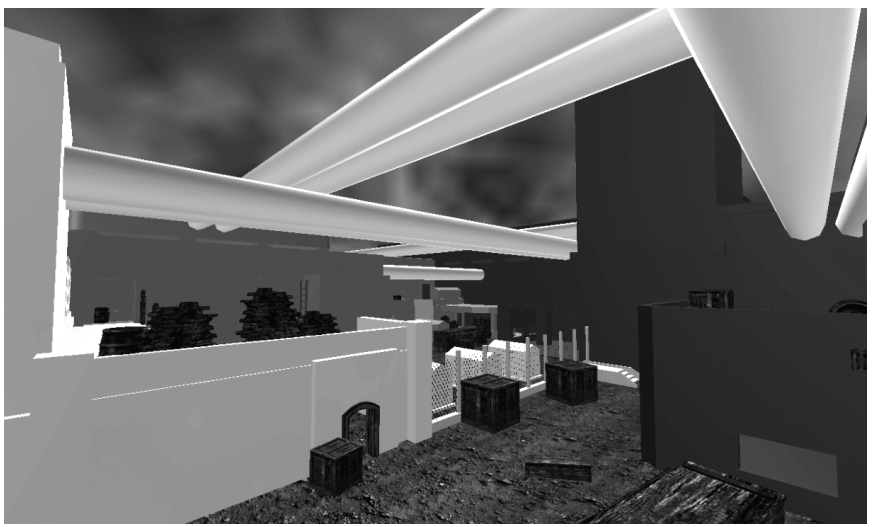
Kiedy mamy już jasną wizję lokacji, możemy wziąć się za tworzenie szkicu 3D. Celem budowania makiety w trójwymiarze jest stworzenie ostatecznego planu przestrzennego lokacji i przetestowanie naszego projektu w grze. Plan taki tworzy się stosunkowo szybko, ponieważ tworzymy go używając brył podstawowych takich jak sześciany, walce, stożki. Na tym etapie naszym celem nie jest stworzenie rzucającej na kolana grafika a sprawdzenie planu lokacji. Również tutaj możemy już wstępnie naszkicować kubaturę budynków, sprawdzić jak bryły komponują się ze sobą, jak z różnych miejsc prezentują się ujęcia panoramiczne itd. Tworzenie takiego wstępnego szkicu lokacji zwane jest też z angielskiego *placeholder-plan* co w swobodnym tłumaczeniu oznacza plan lokacji zbudowany z elementów tymczasowych, które na dalszych etapach pracy są wymieniane i detalizowane.

Szkic 3D możemy wykonać przy użyciu profesjonalnych programów do obróbki grafiki 3D takich jak np. Maya, 3ds Max, czy też darmowego Blendera. Wtedy w nowo utworzonej scenie tworzymy z brył podstawowych zarys lokacji, a następnie eksportujemy ją do edytora i oglądamy w silniku 3D. Najnowsze silniki 3D oferują nam też możliwości wykonania szkicu w edytorze 3D, który w obecnych czasach jest integralną częścią całego silnika 3D. W edytorze 3D często mamy do dyspozycji dodatkowe narzędzia takie jak np. edytor terenu,



Rysunek 4.10. Szkic 3D lokacji wykonany w programie Maya

mieszanie tekstur, predefiniowane elementy otoczenia itd., a to znacznie ułatwia cały proces i skraca czas pracy potrzebny do stworzenia szkicu 3D.



Rysunek 4.11. Szkic 3D lokacji w silniku PainEngine

Należy jednak pamiętać, iż pomimo że jest to wstępny zarys lokacji, musi on zawierać wszystkie ważne elementy architektoniczne takie jak np. schody, windy, drzwi itd. Powinniśmy mieć możliwość swobodnego zwiedzenia całej lokacji i wyłapania ewentualnych błędów i luk w naszej wizji. Głównym celem szkicu

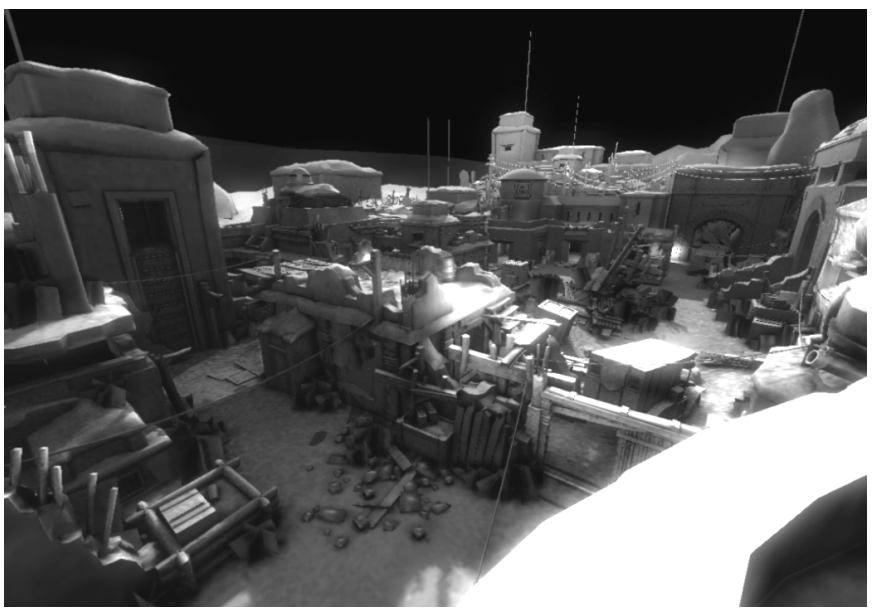
w trójwymiarze jest dopracowanie i kompleksowe przetestowanie planu lokacji - często na etapie pomysłu oraz szkiców koncepcyjnych nie jesteśmy w stanie przewidzieć wszystkiego i dopiero szkic w trójwymiarze ukazuje mocne i słabe strony całej planowanej lokacji. Na tym etapie zaś jesteśmy w stanie stosunkowo szybko nanosić poprawki i dopracować prototyp w pełni, jest to jednocześnie bardzo ważny etap, ponieważ dokonywanie większych zmian na dalszych etapach pracy będzie już bardzo czasochłonne.

Należy także pamiętać, iż nasz szkic 3D posłuży również do przetestowania grywalności i musimy się liczyć z poprawkami i przeróbkami, które będą zgłaszać projektanci rozgrywki lub osoby odpowiedzialne za warstwę techniczną projektu.

4.4 Tworzenie architektury

Gdy kompletny szkic 3D naszej lokacji jest już gotowy, możemy przystąpić do tworzenia docelowej grafiki. W zależności od systemu pracy jaki przyjmiemy, grafikę a szczególnie elementy otoczenie i architekturę możemy tworzyć na dwa sposoby.

Pierwszy sposób polega na tworzeniu lokacji w oparciu o narzędzia dostępne tylko i wyłącznie w edytorze dołączonym do silnika. Najczęściej edytor wyposażony jest w swój własny moduł odpowiedzialny za modelowanie oraz



Rysunek 4.12. Podgląd architektury modeli z silnika PainEngine

teksturowanie brył - przykładem może tu być UnrealED (wykorzystany np. w serii Unreal) lub GTK-Radiant (wykorzystany np. w serii Quake) albo też Hammer (gra Half-Life). W tych edytora przy wykorzystaniu technologii BSP możemy swobodnie tworzyć bryły podstawowe, popularnie zwane brushami i z nich budować całą architekturę świata gry. Dodatkowym wsparciem jest też edytor terenu i roślinności, który umożliwia nam łatwe budowanie lokacji w otwartych krajobrazach. W chwili obecnej jednak odchodzi się od budowania elementów architektury w edytorach na korzyść bardziej dokładnych modeli z profesjonalnych programów 3D, takich jak Maya. Jednak nadal bardzo często wykorzystuje się edytory BSP i terenu przy tworzeniu trójwymiarowych prototypów lokacji i prostej architektury.

Drugi sposób polega na tworzeniu kompleksowych modeli przy wykorzystaniu zewnętrznych programów do obróbki grafiki 3D, takich jak Maya, 3ds Max, Zbrush, a następnie importowanie wcześniej przygotowanych modeli do edytora gry. Tę metodę tworzenia architektury wykorzystuje prawie każdy popularny silnik gier, choć w zależności od systematyki pracy, tworzy się mniejsze lub większe ilości modeli. W silniku PainEngine (który był wykorzystywany przy produkcji gier Painkiller oraz NecroVisioN), całe sceny łącznie z terenem były modelowane i teksturowane w programie Maya a następnie eksportowane do silnika gry, więc można powiedzieć że większość architektury otoczenia powstawała poza edytorem i silnikiem. Dla porównania w serii Quake proporcje te były odwrócone - tam większość modeli otoczenia była tworzona w edytorze

Rysunek 4.13. Podgląd architektury modeli z silnika PainEngine

GTK Radiant. Obecnie zaś dominuje podejście, w którym większość modeli jest tworzona poza edytorem i importowana do silnika (np. Gears of War), choć zazwyczaj są to powtarzalne moduły a nie kompletne sceny. Modele architektury eksportowane z programu 3D do silnika zwane są siatkami statycznymi (ang. *static mesh*) - czyli geometrią, której zazwyczaj nie możemy bezpośrednio modyfikować z poziomu edytora lub samej gry. Muśmy mieć świadomość, iż profesjonalne pakiety do tworzenia grafiki 3D oferują nam nieograniczone wręcz możliwości kształtowania obiektów i jest to główna przyczyna wypierania silnikowych modułów do modelowania.

Rysunek 4.14. Podgląd architektury z teksturami w silniku PainEngine

Dość popularną metodą pracy jest sposób pośredni - czyli tworzenie bazowej architektury w silniku a następnie uszczegóławianie jej modelami wykonanymi w zewnętrznych programach 3D. W chwili obecnej nie ma większego sensu rozwijanie silnikowych programów do modelowania 3D, bo wygodniej jest wykorzystywać już sprawdzone i rozbudowane pakiety takie jak Maya, czy też Blender i eksportować z nich modele do silnika. Jedynie pewne specyficzne elementy, takie jak teren, oświetlenie, materiały, proste animacje, efekty specjalne itd. powstają często bezpośrednio w silniku z użycie różnego rodzaju dedykowanych edytorów np. edytora materiałów, edytora oświetlenia itd.

4.5 Moduły

Jedną z najważniejszych cech, na których opierają się najnowsze silniki jest system modułowego budowania otoczenia. Nie wdając się w szczegóły programistyczne, wszystko sprowadza się do tego, iż tworzymy pojedyncze modele,

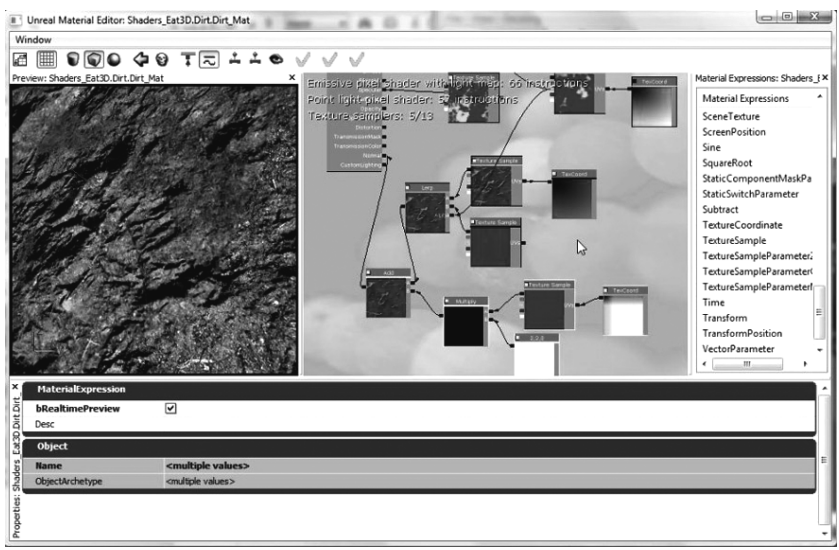
które eksportujemy do silnika. Modele takie są wielokrotnie kopiowane i rozmieszczane w grze jako element otoczenia. Pojedyncze modele możemy także łączyć w większe moduły, czyli grupy modeli, które pozwolą nam szybciej budować np. architekturę.

Posłużymy się tu przykładem, który zobrazuje przebieg prac nad sceną zawierającą kamienicę:

- tworzymy teren w edytorze (podłoże + zarys dróg)
- tworzymy główne bryły budynków z geometrii BSP (ściany + dachy)
- tworzymy dokładne modele elementów wykończeniowych w zewnętrznych programach 3D (drzwi, okiennice itd.)
- detalizujemy architekturę modelami typu static mesh (wstawiamy okna, drzwi itd.) - tworzymy moduły (cały budynek staje się nowym samodzielnym modelem).

4.6 Teksturowanie i materiały

Do gotowych modeli musimy jeszcze przypisać tekstury. Jeśli tworzyliśmy bryły w edytorze, to przeważnie są one automatycznie zmapowane i jedyne co musimy potem zrobić to nałożyć odpowiednie tekstury lub określić liczbę powtórzeń danej tekstury na modelu. Wszystkie te operacje wykonujemy w czasie rzeczywistym, a więc dokładnie widzimy jak zmienia się wygląd obiektu, co znacznie ułatwia prace. Gdy tworzymy modele w programie 3D takim jak Maya lub 3ds Max, to zazwyczaj przy użyciu dostępnych narzędzi musimy sami



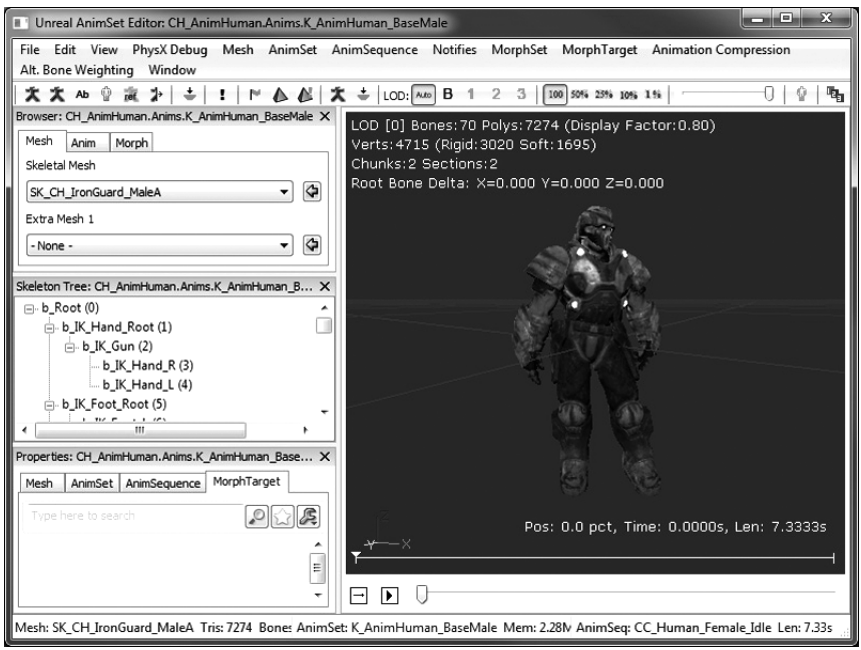
Rysunek 4.15. Edytor materiałów w silniku Unreal

zdefiniować współrzędne mapowania UV. Następnie, aby model stał się składową częścią architektury gry, musimy go jeszcze wyeksportować i zaimportować do silnika, w czym pomagają nam specjalne plug-iny (wtyczki) zapisujące pliki do odpowiednich formatów, np. ASE lub OBJ.

Po zaimportowaniu obiektu w silniku musimy pamiętać o materiałach, które należy zdefiniować, aby model był poprawnie wyświetlany w grze.

4.7 Animacje

Skomplikowane animacje, np. ruch postaci, tworzy się w zewnętrznych programach takich jak Maya czy Motion Builder, warto jednak nadmienić, że proste animacje typu otwieranie drzwi czy okien możemy wykonać korzystając z prostych narzędzi animacyjnych, dostępnych w edytorach silnika gry.



Rysunek 4.16. Edytor animacji w silniku Unreal

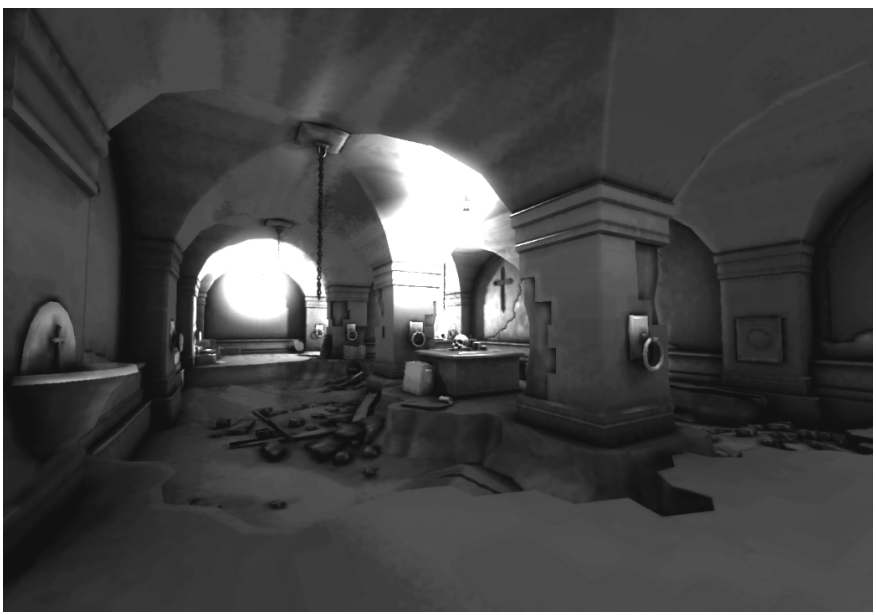
4.8 Oświetlenie

Gdy lokacja jest już cała wymodelowana a tekstury i modele są w wersji finalnej (lub bardzo bliskie wersji ostatecznej), możemy przystąpić do oświetlenia sceny. W większości przypadków będziemy tworzyć oświetlenie wykorzystując

narzędzia dostępne w edytorze silnika i właśnie to środowisko staje się głównym narzędziem pracy oświetleniowca.

Oświetlenie silnikowe możemy podzielić na światła dynamiczne (oświetlające scenę w czasie rzeczywistym) i statyczne (służące do renderowania statycznych map oświetlenia).

Do każdego światła, które dodamy do sceny możemy a wręcz musimy dobrać odpowiedni kolor, zasięg, intensywność i tylko odpowiednie zbalansowanie wszystkich tych parametrów źródeł światła zapewni nam dobre oświetlenie.



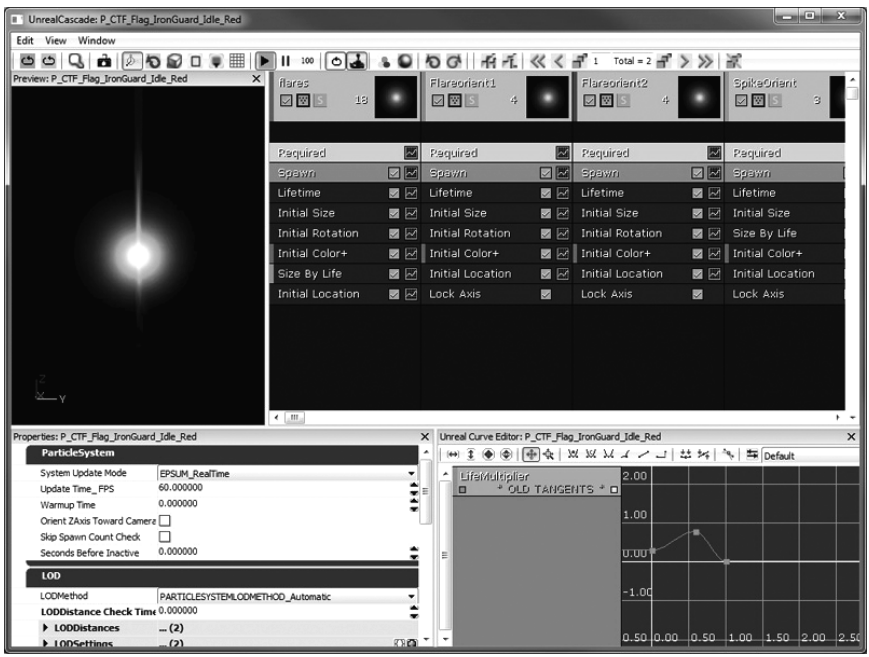
Rysunek 4.17. Podgląd oświetlenia w silniku PainEngine

Tworząc oświetlenie musimy na samym początku założyć jaki chcemy wytworzyć klimat lokacji, a następnie wykorzystując odpowiednio dobrane rodzaje świateł i technik oświetleniowych zrealizować ten zamysł. Pamiętajmy że dobre oświetlenie to połowa sukcesu, to właśnie gra świateł i cieni pomoże nam wydobyć bryłę architektoniczną, pogłębić nastrój grozy lub sielanki itd.

4.9 Efekty specjalne

Doskonałym dopełnieniem atmosfery na lokacji są efekty specjalne, zazwyczaj tworzone w oparciu o systemy cząsteczkowe (ang. particles) za pomocą specjalnego edytora, który jest integralną częścią większości silników 3D. Dzięki cząsteczkom możemy dodawać takie efekty jak: ogień, iskry, dym ale także

deszcz, błyskawice i wszelkiej maści inne zjawiska atmosferyczne. Specyficznym dopełnieniem efektów specjalnych jest efekt delikatnego zamglenia albo mocnej mgły, który przy odpowiednio dobranych ustawieniach pozwala pogłębić wrażenie przestrzenności i kubaturę brył architektonicznych.



Rysunek 4.18. Edytor efektów specjalnych w silniku Unreal

Osobną grupą efektów, która ostatnimi laty stała się nieodłączną częścią wyglądu gier, są tzw. *post-effects* - czyli efekty postprocessingu, do który zaliczamy między innymi głębię ostrości (*depth of field*), rozmycia w ruchu (*motion blur*), balansy bieli i kolorów (*color balances*) itd. Korzystanie z efektów postprocesowych pozwala nam jeszcze precyzyjniej dookreślić i zbalansować wszelkie efekty specjalne i nadać lokacji niepowtarzalny charakter.

4.10 Optymalizacja

Proces optymalizacji zazwyczaj trwa nieustannie podczas wszystkich etapów tworzenia lokacji. Na bieżąco optymalizujemy bryły, światła i tekstury. W sceneriach, które wyświetlane są w czasie rzeczywistym, płynność animacji jest jednym z kluczowych elementem odbioru (szczególnie w grach). Każdy silnik ma swoje ograniczenia wydajnościowe i naszym zadaniem jest jak najlepsze

zbalansowanie jakości z wydajnością, tak by obraz wyświetlany na ekranie monitora lub telewizora był gładki i płynny.

Rysunek 4.19. Ostateczny wygląd lokacji wyświetlany w czasie rzeczywistym w silniku 3D

Rysunek 4.20. Ostateczny wygląd lokacji wyświetlany w czasie rzeczywistym w silniku 3D

Optymalizacja obejmuje: geometrię (optymalizacja siatek modeli), tekstury (optymalizacja rozdzielczości, kompresji, mapowania), oświetlenie (zasięgi świateł i ustawienia cieni obiektów), materiały (ilość i złożoność materiałów używanych przez modele). Dobre i rzetelne wykonywanie zadań związanych z optymalizacją pozwala nam na dodanie większej ilości obiektów, efektów, świateł przy zachowaniu płynnego wyświetlania obrazu w odpowiedniej liczbie klatek na sekundę. W ten sposób nasza lokacja staje się znacznie bogatsza i jeszcze atrakcyjniejsza w sferze audiowizualnej.

5

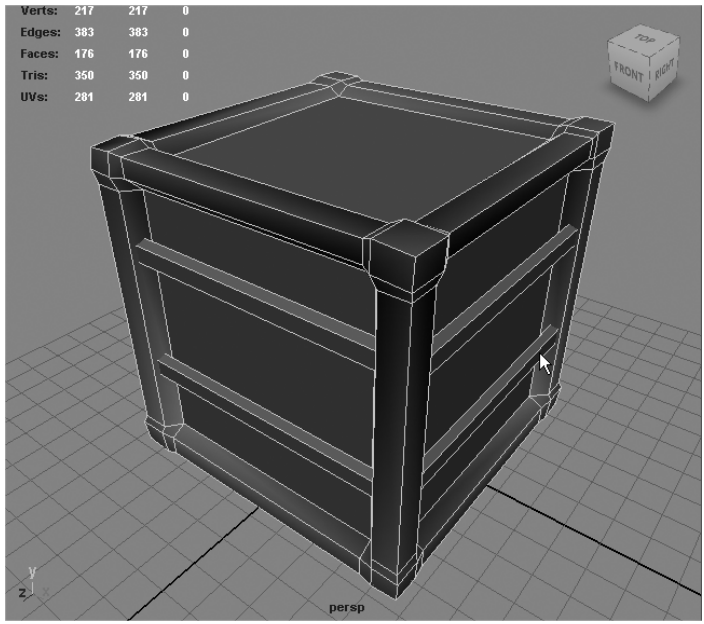
Tworzenie fotorealistycznych tekstur - wprowadzenie

Do tej pory zajmowaliśmy się głównie zagadnieniem modelowania, czyli tworzenia siatek odwzorowujących kształty trójwymiarowych obiektów. Modelowanie jest jednak tylko pierwszym krokiem w stronę uzyskania przekonującej sceny 3D. Aby modele nabrały odpowiedniego - realistycznego lub stylizowanego - charakteru, konieczne jest zastosowanie dobrze przygotowanych tekstur i materiałów.

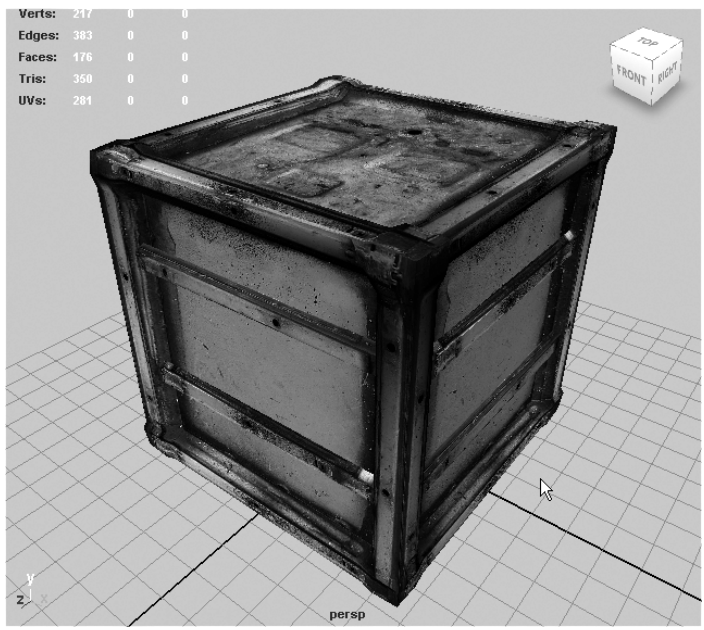
Tekstur w postaci obrazów 2D, zarówno ręcznie malowanych jak i fotograficznych, używamy w niemal wszystkich obszarach zastosowań grafiki trójwymiarowej, jednak w żadnej dziedzinie nie mają one tak wielkiego znaczenia jak w aplikacjach interaktywnych.

Ze względu na wymaganą szybkość odświeżania sceny w czasie rzeczywistym i płynność animacji, silniki gier i narzędzia do wizualizacji narzucają duże ograniczenia w zakresie dokładności modeli i materiałów. Najczęściej prowadzi to do sytuacji, w których nie możemy pozwolić sobie na tworzenie zadowalająco szczegółowych modeli i skomplikowanych, wielowarstwowych shaderów, jakie moglibyśmy stosować w statycznych obrazach lub też renderowanych filmach. Pomimo ciągłego rozwoju układów wspomagających wyświetlanie grafiki 3D w czasie rzeczywistym, praktycznie zawsze podlegamy ograniczeniom dotyczącym liczby ścianek w obiektach. Oznacza to, że jeśli chcemy uzyskać wiarygodnie wyglądający model, jedynym wyjściem pozostaje stworzenie odpowiednich tekstur, które dodadzą do niego szczegóły nie nadające się do wymodelowania na poziomie siatki oraz zamaskują fakt kanciastości bryły związany z uproszczeniami geometrii.

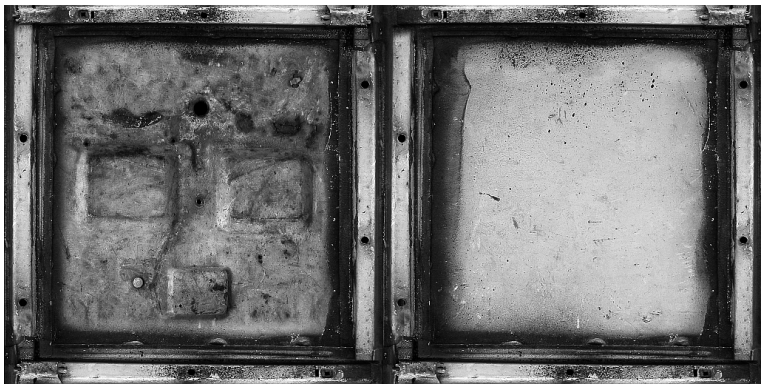
Rysunek 5.1 ilustruje obiekt, który zbudowano z małej liczby ścianek (350 trójkątów) i którego siatka wygląda bardzo prymitywnie. Na rysunku 5.2 można obejrzeć ten sam obiekt po nałożeniu na niego odpowiednio spreparowanej tekstury fotograficznej (rysunek 5.3). Tekstura ta maskuje ostrość krawędzi modelu i dodaje do niego szereg szczegółów czyniących go realistycznym. Warto zwrócić uwagę na to, że materiał nałożony na powierzchnię modelu jest bardzo prosty i nie wykorzystuje żadnych specyficznych efektów - jedynym jego składnikiem jest fotografia użyta jako tekstura koloru.



Rysunek 5.1. Model o prymitywnej siatce wygląda mało realistycznie



Rysunek 5.2. Model z rysunku 5.1 po nałożeniu tekstury uzyskuje realistyczny wygląd



Rysunek 5.3. Fotograficzna tekstura wykorzystana do modelu skrzyni

5.1 Tekstury a materiały

W grafice 3D (nie tylko interaktywnej) potocznie pojęcia tekstur i materiałów często są stosowane wymiennie, co prowadzi do licznych nieporozumień, dlatego warto poświęcić tu chwilę na uporządkowanie tych zagadnień.

Materiał jest to funkcja programu graficznego lub silnika gier opisująca właściwości powierzchni danego obiektu. W zależności od tego, jakie możliwości ma dany system do wyświetlania grafiki 3D, za pomocą materiału określamy między innymi takie właściwości jak:

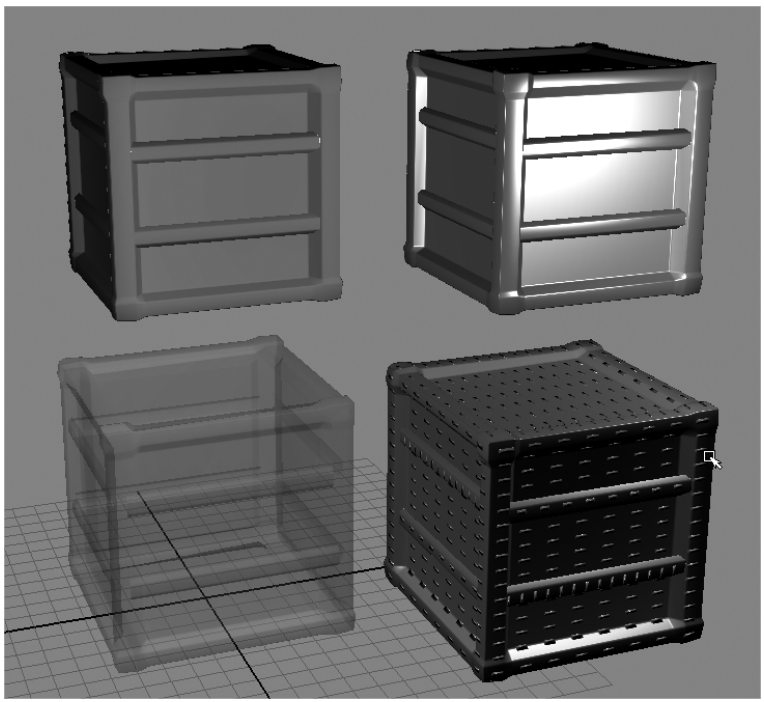
- barwa światła rozpraszanego i odbijanego
- model oświetlenia (matematyczny model interakcji światła z powierzchnią)
- faktura powierzchni
- intensywność, kolor i rozmycie odblasków
- przezroczystość, iluminacja, odbicia lustrzane otoczenia i inne.

Tekstura jest to obraz 2D (najczęściej w postaci bitmapy czyli pliku z rozszerzeniem BMP, TGA, JPG itp), używany do sterowania tymi właściwościami na powierzchni obiektu. W najprostszym przypadku, przedstawionym wcześniej na rysunku 5.1, piksele tekstury określają kolor obiektu. Rzutowanie tekstury na obiekt w celu określenia właściwości materiału nazywamy *mapowaniem*. Oprócz tekstur bitmapowych w grafice 3D stosuje się też czasem generowane za pomocą specjalnych algorytmów tzw. tekstury proceduralne, jednak w grafice interaktywnej są one mniej przydatne ze względu na obciążanie zasobów systemowych podczas renderingu.

Na rysunku 5.4. przedstawiony jest ten sam model 3D z kilkoma przykładowymi materiałami o różnie dobranych właściwościach (za wyjątkiem ostatniego przykładu nie wykorzystano tu żadnych tekstur):

- model oświetlenia *Lambert* dający matową powierzchnię o szarym kolorze (domyślny materiał w programie Maya)

- model oświetlenia *Phong* uwzględniający odblaski światła
- materiał półprzezroczysty
- materiał z mapą nierówności (bump-mapping) dającą efekt faktury na powierzchni

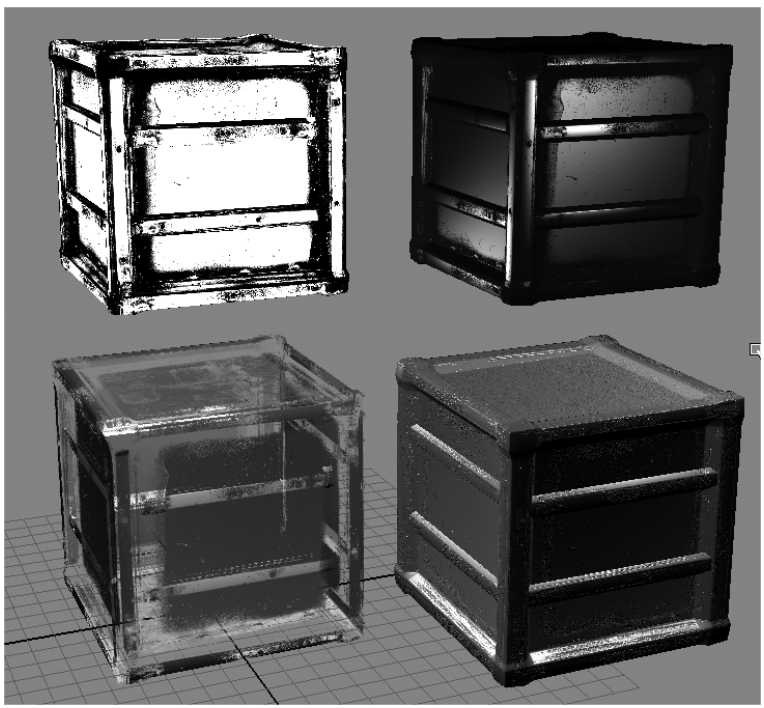


Rysunek 5.4. Przykłady materiałów na modelu skrzyni

Model z rysunku 5.4 posiada właściwości obiektu zdefiniowane jednolicie dla całej powierzchni (wyjątkiem jest ostatni przypadek, gdzie mapa nierówności daje efekt lokalnych uwypukleń w specyficznych miejscach, jednak także i tutaj efekt jest rozłożony równomiernie na całym obiekcie). Większość tych właściwości, podobnie jak barwę, można zdefiniować w sposób zróżnicowany dla poszczególnych fragmentów obiektu, wykorzystując do tego celu specjalnie przygotowane tekstury. W tym przypadku decydują one o intensywności danego efektu analogicznie jak tekstura koloru decyduje o składowych RGB w danym punkcie obiektu. O ile kolor pobierany jest z trzech składowych bitmapy (czerwonej, zielonej i niebieskiej), o tyle inne efekty mapowane teksturami najczęściej wykorzystują pojedynczy kanał albo uśrednioną luminancję obrazu, czyli traktują obraz tak, jakby był zapisany w skali szarości. Jasnym pikselom odpowiadają wtedy wysokie intensywności danego efektu, natomiast ciemnym - niskie.

Przykłady mapowania różnych efektów za pomocą tekstury można obejrzeć na rysunku 5.5, są to kolejno:

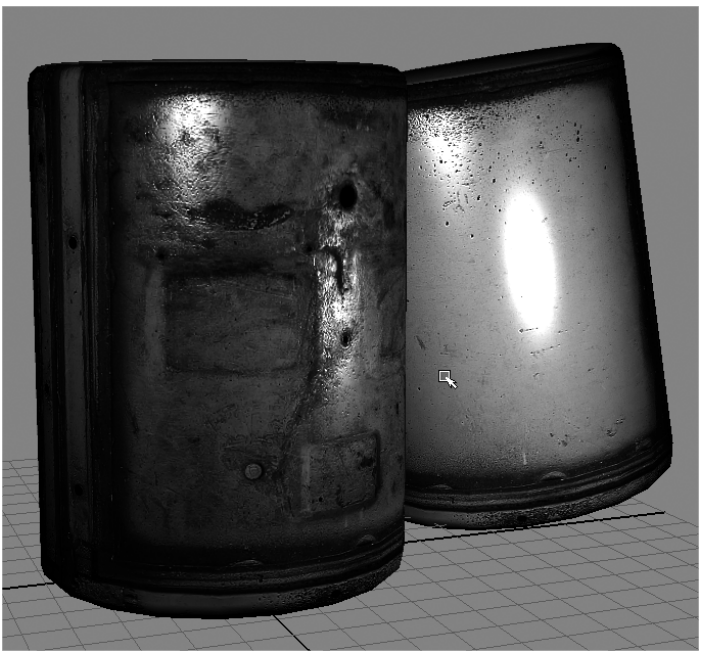
- mapowanie efektu samoświecenia obiektu - jasne piksele tekstury reprezentują miejsca, z których obiekt emituje światło, ciemne piksele odpowiadają miejscom nie emitującym światła - obiekt wygląda jakby sam był źródłem światła i nie działa na niego oświetlenie sceny.
- mapowanie poziomu odblasków - jasne piksele to miejsca gdzie na obiekcie pojawiają się odblaski, ciemne piksele to obszary nie odbijające światła
- mapowanie przezroczystości - jasne piksele są to obszary wypełnione (nieprzezroczyste), natomiast ciemne piksele to niewidoczne (przezroczyste lub półprzezroczyste) fragmenty powierzchni
- mapowanie nierówności (bump-mapping) - jasne piksele to wypukłości, ciemne to wgłębienia w powierzchni obiektu



Rysunek 5.5. Przykłady mapowania różnych efektów materiału za pomocą tej samej tekstury

Jeszcze kilka-kilkanaście lat temu materiały w grafice interaktywnej pozwalały jedynie na mapowanie bardzo prostych efektów, takich jak barwa, za pomocą pojedynczych tekstur. Obecnie wiele silników gier umożliwia tworzenie dość rozbudowanych materiałów, uwzględniających złożenie powyższych

i innych efektów w jednym shaderze (rysunek 5.6). Należy jednak pamiętać, że w wielu przypadkach ze względu na wydajność aplikacji nie powinniśmy stosować zbyt złożonych materiałów, nawet jeśli jest to technicznie możliwe.



Rysunek 5.6. Złożony materiał uwzględniający mapowanie koloru, nierówności i odblasków

Bardzo często na dobór materiałów ma wpływ docelowa platforma, na której uruchamiana będzie gra czy aplikacja. Zaawansowane silniki graficzne, takie jak Unreal czy Vision dobrze radzą sobie ze złożonymi materiałami nawet w rozbudowanych scenach, ale to przy założeniu, że tworzymy obiekty z myślą o wydajnych platformach sprzętowych takich jak komputery PC czy konsole stacjonarne. Jeśli jednak przygotowujemy grę lub prezentację pod kątem telefonów komórkowych lub przeglądarek internetowych, często musimy ograniczać się wyłącznie do stosowania tekstur koloru lub przezroczystości w stosunkowo małych rozdzielczościach, podobnie jak w grach na PC-ty czy konsole kilka lat temu.

5.2 Rozdzielczość i liczba tekstur a wydajność aplikacji

Rozdzielczość obrazu wykorzystanego jako tekstura ma ogromny wpływ na wygląd materiału i możliwość przedstawiania obiektu w zbliżeniach. Dlatego

biorąc pod uwagę wyłącznie kryterium jakości wizualnej, można przyjąć, że im wyższa rozdzielczość tekstury, tym lepszy efekt końcowy. W epoce wydajnych komputerów PC i tanich aparatów cyfrowych samo tworzenie tekstur o wysokich rozdzielczościach nie jest problemem technicznym, jednak tworzenie grafiki interaktywnej wiąże się z koniecznością spełniania wymagań narzuconych przez platformę sprzętową, z tego zaś wynika potrzeba stałego kontrolowania ilości danych wykorzystanych w scenie.

Praktycznie na każdej platformie sprzętowej dane aplikacji nie mogą przekraczać pewnej objętości pamięci, a kolekcje tekstur są zazwyczaj największymi objętościowo zbiorami danych w grze lub prezentacji interaktywnej. Na przykład nieskompresowana tekstura RGB w rozdzielczości 1024×1024 zajmuje 3MB na dysku i w pamięci operacyjnej - jest to więcej, niż niektóre platformy mobilne dopuszczają jako całkowita objętość grafiki w danej aplikacji.

Ograniczenia, które musimy zastosować, związane są z różnymi czynnikami, jednak podstawowym jest konieczność przechowywania danych graficznych w obszarze szybkiej i bezpośrednio dostępnej dla procesora pamięci RAM. Scena w grafice interaktywnej podlega dynamicznym zmianom i często, w przeciwieństwie do statycznych obrazów lub filmów, nie jesteśmy w stanie przewidzieć jaka część sceny będzie w danej chwili wyświetlona - a to z kolei oznacza, że system graficzny musi mieć nieprzerwany dostęp do wszystkich zasobów by płynnie odświeżać widok sceny kilkadziesiąt razy na sekundę. Stąd musimy pamiętać o następujących limitach:

- Na komputerach PC ogranicza nas pamięć RAM znajdująca się na karcie graficznej - popularne karty mają dziś od 256 do 2048 MB pamięci i tworząc aplikacje musimy przystosować się do minimalnej wymaganej konfiguracji sprzętowej. Jeśli dane graficzne wymagane do wyświetlenia sceny przekraczają objętość pamięci RAM, konieczne jest doładowywanie ich z dysku co zazwyczaj prowadzi do obniżenia wydajności i płynności aplikacji. Dotyczy to nie tylko modeli, tekstur i animacji, ale innych danych takich jak np. bufory renderingu, które również muszą być przechowywane w pamięci RAM i ograniczają dostępny obszar dla obiektów graficznych.
- Na stacjonarnych konsolach gier aktualnej generacji - Xbox 360, PlayStation 3 - całkowita pojemność pamięci jest ograniczona systemowo do, odpowiednio, 256 i 512MB i w pamięci tej muszą zmieścić się zarówno dane graficzne jak i pozostałe dane aplikacji. Do tego ograniczeniem są rozmiary całej aplikacji narzucone nośnikiem lub dopuszczalną objętością pliku z grą w przypadku aplikacji pobieranych z sieci. W przypadku konsol opartych na starszych rozwiązaniach technologicznych (Wii) dostępne jest maksymalnie kilkadziesiąt megabajtów pamięci RAM.
- Platformy i konsole mobilne, w tym także telefony komórkowe, mają zazwyczaj jeszcze mniej pamięci operacyjnej i w tym przypadku całość aplikacji (kod wykonywalny, obiekty, tekstury, dźwięki, muzyka, tekst itd) musi obejmować nie więcej niż kilka, kilkanaście lub kilkadziesiąt megabajtów. Tendencja ta się zmienia - platformy przenośne są coraz wydajniejsze, ale

wciąż ich zasoby podlegają znacznie większym ograniczeniom niż w przypadku PC-tów lub konsol stacjonarnych.

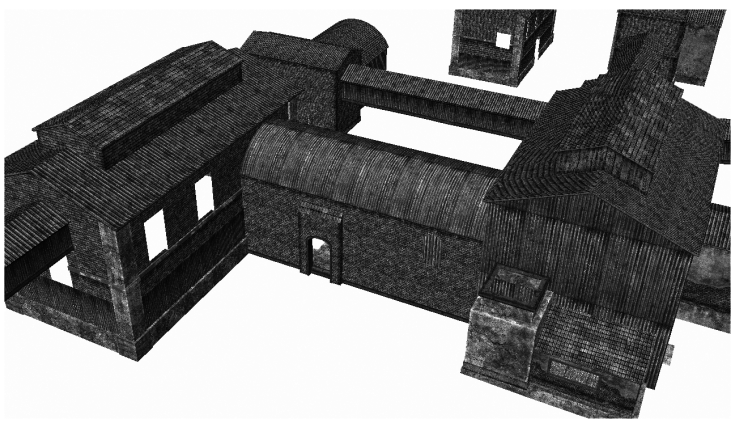
- Aplikacje wyświetlane w przeglądarkach internetowych mogą wydawać się ograniczone jedynie pojemnością serwera danych, jednak trzeba pamiętać, że wszystkie dane muszą zostać przesłane przez sieć. Aby uniknąć zmuszania odbiorcy do długiego czekania na uruchomienie aplikacji, w tym przypadku musimy ograniczać objętość danych ze względu na zakładaną minimalną szybkość transferu oraz konieczność zapewniania zgodności z komputerami o słabszej wydajności, na których zazwyczaj taka aplikacja powinna działać.

Szczególnie w przypadku tekstur łatwo jest o przekroczenie zakładanego zapotrzebowania na zasoby, ponieważ typowy plik tekstury w rozdzielczości 1024×1024 w zależności od formatu zajmuje nawet do kilku megabajtów, zaś w celu uzyskania urozmaiconej sceny potrzebne jest zazwyczaj od kilkudziesięciu do kilkuset tekstur. Dlatego wiele spośród omawianych w tym rozdziale i w całej książce zagadnień uwzględnia konieczność stosowania tekstur w taki sposób by uzyskać jak najlepsze efekty wizualne przy minimalnej ich liczbie i rozdzielczości.

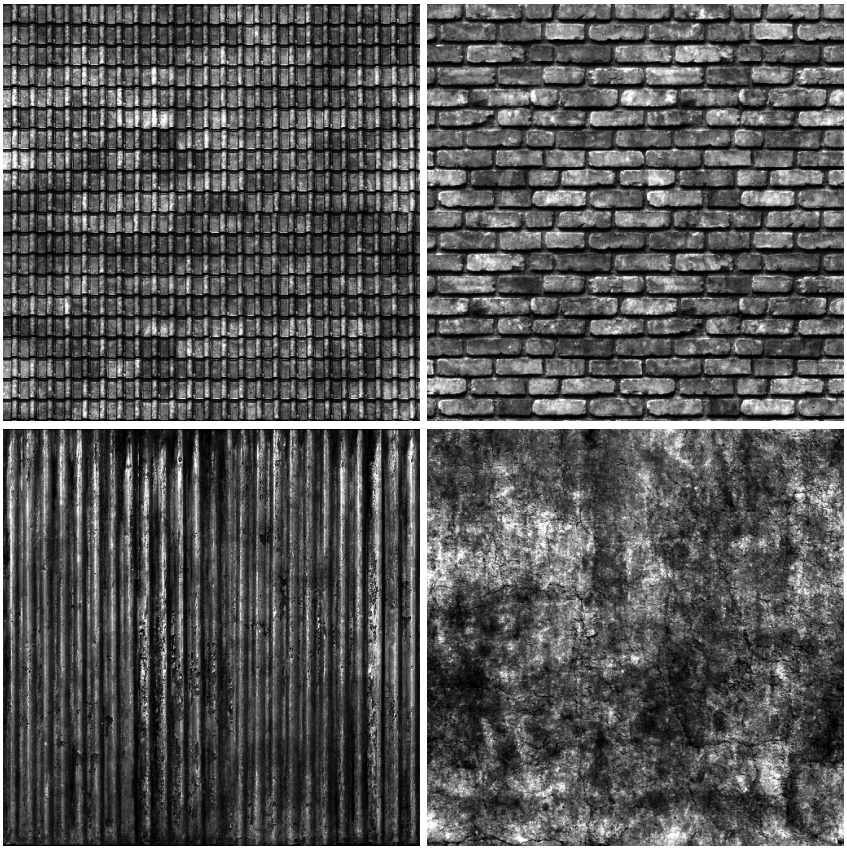
Poniżej omawiamy najważniejsze metody optymalizacji wykorzystania tekstur w grafice interaktywnej.

5.3 Korzystanie z zapętlonych (powtarzalnych) tekstur

Tekstury powtarzalne to takie, które można powtórzyć wiele razy na powierzchni danego obiektu i tym samym ograniczyć rozdzielczość pojedynczego obrazka. Rysunek 5.7 przedstawia zespół budynków, do których otekstrowania



Rysunek 5.7. Zespół wielu budynków został otekstrowany przy wykorzystaniu kilku powtarzalnych tekstur, przedstawiających cegły, beton, blachę i dachówki



Rysunek 5.8. Większość powierzchni w całej scenie może korzystać z kilku powtarzających się tekstur

wykorzystano głównie dwie tekstury - ceglany mur i blachę falistą, mapowane na różnych fragmentach ścian i dachów wszystkich budynków. Pozostałe powierzchnie pokryto dodatkowymi teksturami, takimi jak beton czy dachówki (rysunek 5.8). Wszystkie te tekstury są zapętlone, czyli mogą być powtarzane na każdej ściance obiektu bez widocznych szwów i dać efekt długiej, jednolitej powierzchni. Dzięki temu wzór przedstawiających stosunkowo niewielką liczbę cegieł w zupełności wystarcza do pokrycia dowolnie dużej ściany.

5.4 Wykorzystywanie tych samych tekstur na różnych obiektach w scenie

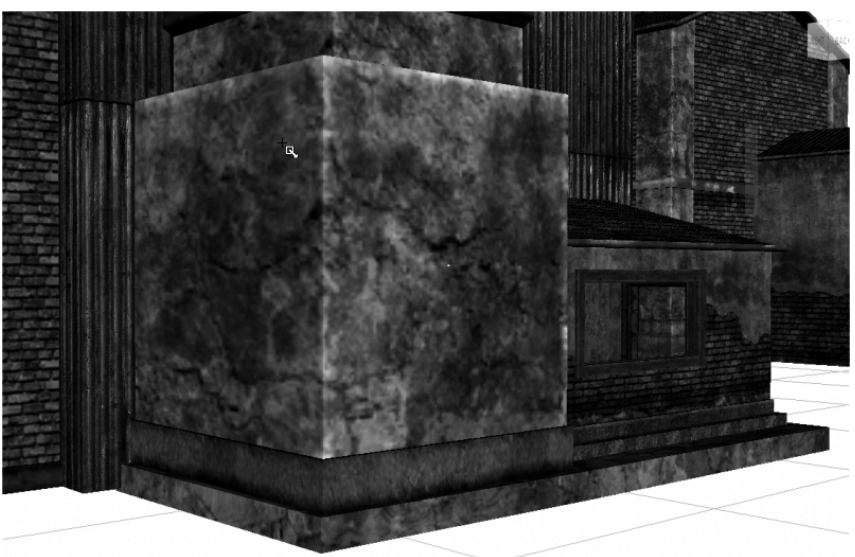
W wielu przypadkach możemy zrezygnować z tworzenia nowych tekstur dla każdego kolejnego obiektu wstawianego do sceny. W przypadku modeli o podo-

bnej strukturze powierzchni, warto stosować te same tekstury. Na rysunku 5.8 pokazano tekstury powtarzające się na wszystkich budynkach z rysunku 5.7 - do oteksturowania podobnych powierzchni w całej scenie wystarczył pojedynczy wzór cegieł, betonu, blachy itp.

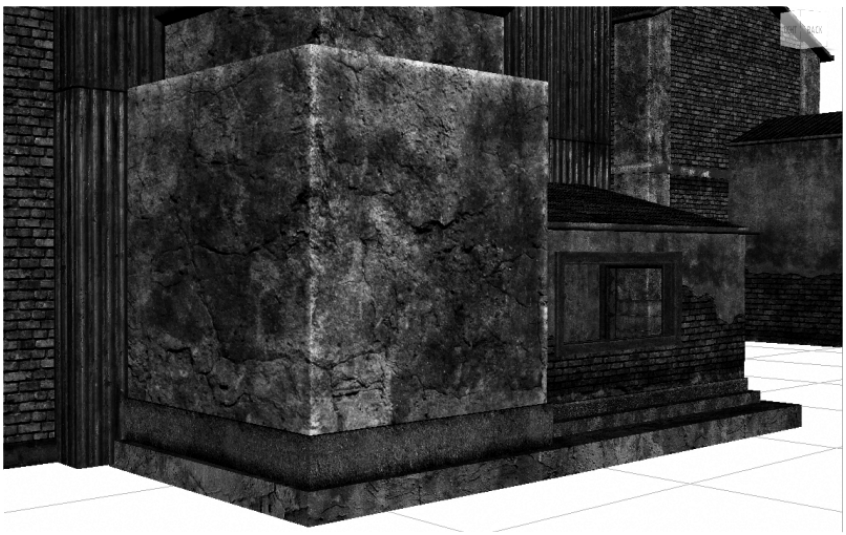
5.5 Optymalna rozdzielczość tekstur

W ogromnej większości przypadków dążymy do tego, by tekstury były jak najbardziej ostre i wyraziste, czyli staramy się uniknąć efektów rozmycia powierzchni spowodowanych zbyt niską rozdzielczością tekstur. Rysunki 5.9 i 5.10 przedstawiają fragment ściany ze zbyt niską i z odpowiednio wysoką rozdzielczością tekstury betonu.

W zależności od tego, gdzie obiekt z daną teksturą ulokowany jest w świecie gry, bardzo często możemy odgórnie ograniczyć rozdzielczość tekstur bez strat jakości. Podstawowym wskaźnikiem dla doboru odpowiedniej rozdzielczości tekstury jest rozmiar, w jakim ma być wyświetlany dany obiekt na ekranie. Przy tworzeniu każdej tekstury należy - o ile tylko jest to możliwe - ocenić w jakim stopniu będzie ona wypełniała kadr obrazu przy największym możliwym zbliżeniu obiektu. Jeśli obiekt ma być wyświetlany z daleka, nie ma sensu stosować dla niego szczegółowych i dużych tekstur. Gdy zaś obiekt ma być pokazywany na pierwszym planie, warto zarezerwować dla niego więcej miejsca w pamięci na odpowiednio dokładne tekstury.



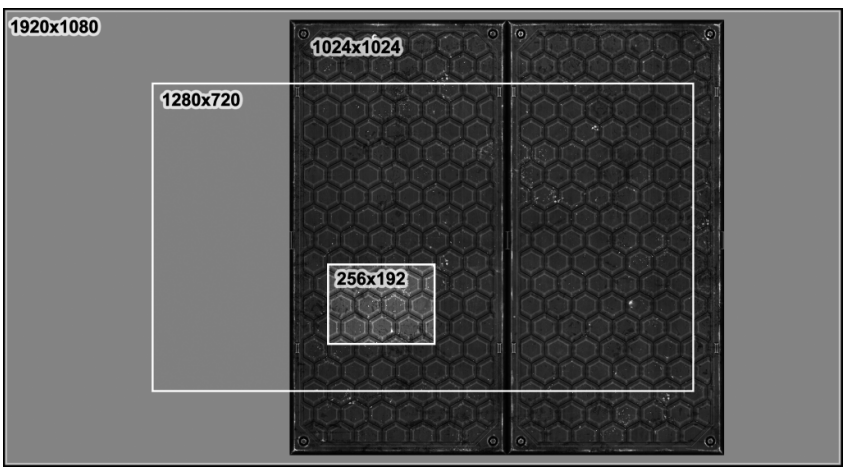
Rysunek 5.9. Powierzchnia betonowej ściany wygląda na rozmazaną, co jest efektem wykorzystania tekstury w zbyt niskiej rozdzielczości



Rysunek 5.10. Aby powierzchnia ściany była ostra i wyraźna, tekstura musi posiadać odpowiednio wysoką rozdzielczość

Należy przy tym uwzględnić docelową rozdzielczość obrazu i na jej podstawie oszacować optymalną rozdzielczość tekstury (rysunek 5.11). Przykłady:

- W grze FPP (first person perspective, czyli świat pokazywany z oczu gracza) gracz ma możliwość podejścia bezpośrednio do ściany budynku w taki



Rysunek 5.11. Tekstura w rozdzielczości 1024×1024 piksele wyświetlana w skali 1:1 na tle ekranów o różnych rozdzielczościach

sposób, że ściana ta w całości wypełnia wysokość kadru. Jeśli gra jest przeznaczona na konsolę wyświetlającą obraz w rozdzielczości 1280×720, to tekstura 512×512 dla takiej ściany może okazać się zbyt mała, z kolei tekstura 2048×2048 byłaby zbyt duża gdyż i tak piksele obrazu w tej rozdzielczości nie byłyby w stanie oddać wszystkich szczegółów takiej tekstury. Jeśli tylko ograniczenia pamięci nie zmuszają nas do intensywniejszej optymalizacji, najrozsądniej byłoby przyjąć rozdzielczość 1024×1024 dla tekstury ściany.

- W grze TPP (third person perspective, czyli świat pokazywany z kamery umieszczonej nad lub za graczem) ze względu na ustawienie kamery kwiaty oraz małe kamienie na podłożu wypełniają nie więcej niż po 10% wysokości i szerokości kadru. Jeśli gra wyświetlana jest w standardzie HD 1920×1080, to pojedynczy kamień czy roślina nie będzie miała większego rozmiaru niż ok. 200×100 pikseli. To oznacza, że docelowo można zastosować dla tych obiektów tekstury 256×256 lub nawet mniejsze, jednakże tekstura terenu wypełniającą połowę kadru powinna mieć już rozdzielczość 1024×1024 (zobacz rysunek 5.11).
- W aplikacji na telefony komórkowe o rozdzielczości 256×192 budynki wypełniają maksymalnie połowę kadru. Można więc założyć, że rozdzielczość tekstur nie musi być większa niż 128×128 pikseli.

Podsumowując, aby dobrać odpowiednią rozdzielczość tekstur, należy uwzględnić następujące czynniki dla danej platformy sprzętowej i aplikacji:

- Ilość całkowitej pamięci dostępnej na tekstury
- Największa dopuszczalna rozdzielczość lub rozmiar pliku dla pojedynczej tekstury
- Docelowa rozdzielczość ekranu
- Maksymalne powiększenie danej tekstury jakie może wystąpić na ekranie w trakcie działania aplikacji
- Sposób zarządzania danymi graficznymi przez aplikację - niektóre systemy i silniki gier umożliwiają dynamiczne ładowanie oraz usuwanie tekstur z pamięci w trakcie działania aplikacji, inne wymagają tego, aby wszystkie tekstury dla danej sceny znajdowały się przez cały czas w pamięci
- Poziom szczegółowości tekstur - jeśli nie planujemy dodawać do nich wielu drobnych szczegółów, można dla poprawienia wydajności stosować tekstury w małej rozdzielczości
- Liczba dodatkowych efektów materiału mapowanych za pomocą tekstur (normal, specular, bump, opacity itd.), które często wymagają stosowania dodatkowych tekstur w odpowiednio wysokiej rozdzielczości

W przypadku komputerów PC, których karty graficzne z roku na rok pozwalają wyświetlać obraz w coraz większej rozdzielczości i mają coraz więcej pamięci na tekstury, powyższe założenia spełnia się poprzez ustanowienie tzw. minimalnych wymagań sprzętowych, czyli określenie najsłabszej dopuszczalnej konfiguracji komputera, przy której aplikacja będzie działała wystarczająco płyn-

nie. Innymi słowy, z góry zakłada się, że do działania programu konieczne jest posiadanie komputera o pewnych parametrach lub lepszego, a na platformach nie spełniających tych wymagań aplikacja może w ogóle nie działać.

Aby uzyskać efekt odpowiedniej szczegółowości powierzchni w sytuacjach, w których nie możemy pozwolić sobie na stosowanie tekstur o wysokiej rozdzielczości, czasem wykorzystuje się tak zwany detail-mapping czyli nałożenie na podstawową teksturę obiektu drugiej tekstury o stosunkowo małej rozdzielczości lecz dużo większym zagęszczeniu mapowania. Dzięki temu symulujemy drobnoziarniste szczegóły powierzchni nie zwiększając zbyt mocno ogólnego zapotrzebowania na pamięć tekstur (jedną detail-mapę, np. ziarnistość betonu, można wykorzystać na wielu teksturach ścian). Technika ta wymaga jednak stosowania bardziej złożonych materiałów, które same w sobie mogą stanowić obciążenie dla systemu.

5.6 Formaty i kompresja tekstur

Tekstury w ogromnej większości przypadków są zwykłymi obrazami rastrowymi (bitmapowymi), dlatego teoretycznie można przechowywać je w dowolnych formatach plików, takich jak BMP, TIF, GIF czy JPG, jednak ze względów praktycznych kilka wybranych formatów stanowi standardy w zapisywaniu tekstur do silników graficznych. Nawet gdy dany silnik 3D obsługuje wiele różnych formatów, warto ograniczyć się do tych, które są optymalne dla grafiki interaktywnej.

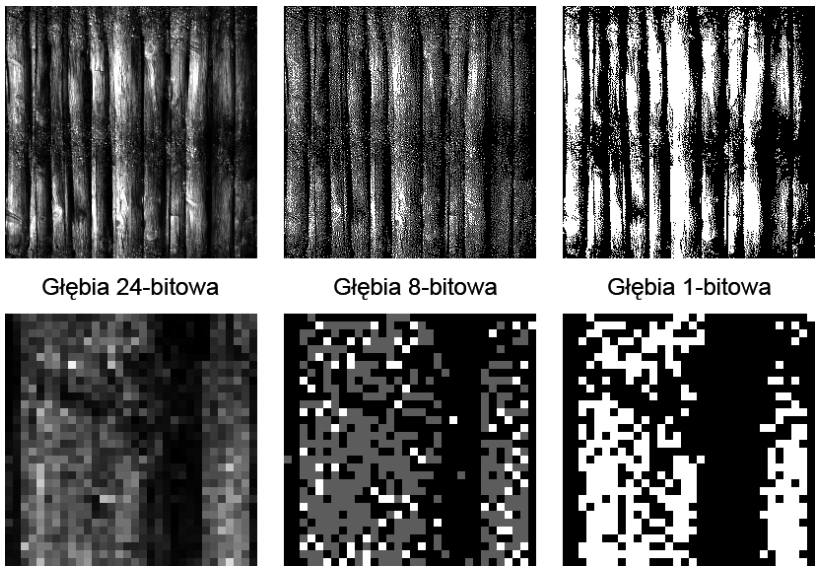
Najczęściej format plików z teksturami jest nam z góry narzucony poprzez technologię, na której oparta jest dana aplikacja, jednak niektóre technologie obsługują więcej niż 1 format plików rastrowych, należy wtedy dodatkowo zdecydować co w danym przypadku jest optymalne dla poszczególnych tekstur.

5.7 Głębia bitowa obrazu

Ważnym czynnikiem przy doborze formatu i sposobu zapisu plików jest głębia bitowa obrazu, czyli parametr określający liczbę bitów przechowujących informację o składowych koloru każdego piksela. Wartość ta określa liczbę kolorów jakie mogą występować w obrębie tekstury a także ewentualne dodatkowe informacje poza kolorem, które można zapisywać w teksturze. Na rysunku 5.12 pokazano przykładową teksturę i jej wycinek w powiększeniu, zapisaną w różnych głębiach bitowych.

W grafice interaktywnej najczęściej tworzy się tekstury o następującej głębi bitowej:

- 1 bit - umożliwia zapisywanie dwóch wartości koloru (czerni i bieli). Tego typu tekstura rzadko bywa stosowana do określania barwy obiektu, jednakże często można ją wykorzystywać do określania przezroczystości mate-

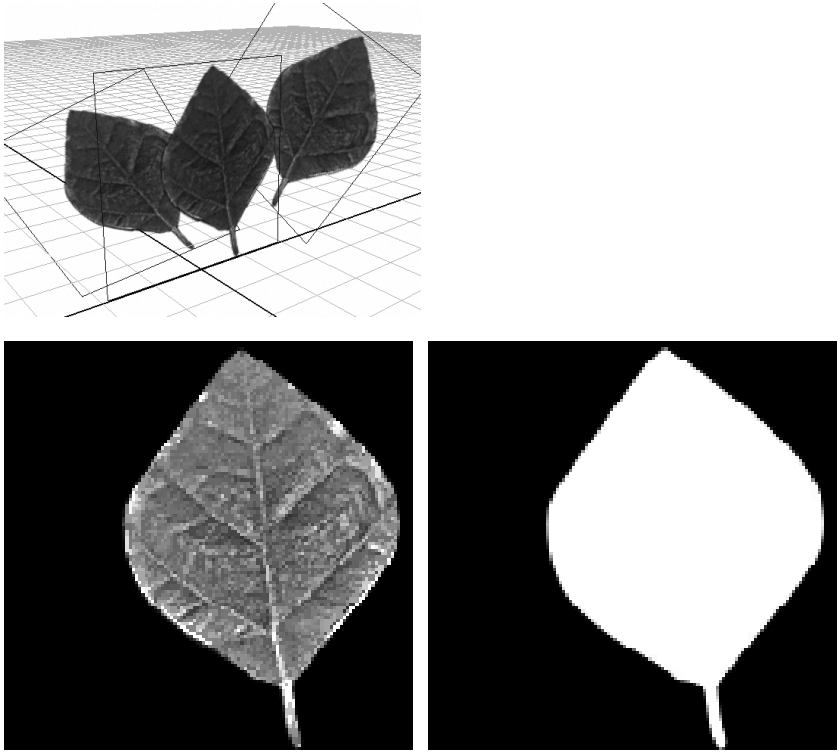


Rysunek 5.12. Tekstura 256×256 pikseli zapisana w różnych głębiach bitowych (u góry) i jej wycinek 32×32 piksele w powiększeniu (u dołu)

riału (gdzie czarny kolor zazwyczaj odpowiada przezroczystym pikselom, a biały nieprzezroczystym).

- 8 bitów - pozwala określić kolor piksela pobierany z tabeli liczącej 256 pozycji. W przypadku 8-bitowej głębi koloru istotny jest odpowiedni dobór palety - w zależności od przeznaczenia tekstury może to być skala 256 poziomów szarości (np. do określania poziomu odblasków materiału albo stopnia przezroczystości) lub wybrana paleta barw (dla tekstur koloru). Głębi 8-bitowej używamy wtedy gdy nie jest nam potrzebna dokładna informacja o kolorze lub gdy tworzymy aplikację na platformę z mocno ograniczonymi zasobami pamięci graficznej (np. konsole przenośne).
- 24 bity (RGB) - w palecie 24-bitowej możemy zapisać ponad 16 milionów kolorów (3 kanały: R, G i B, każdy opisany 8 bitami) i jest to jeden z najpopularniejszych formatów zapisu grafiki rastrowej, nie tylko w aplikacjach interaktywnych.
- 32-bity (RGBA) - w tym przypadku pierwsze 24 bity służą do opisu składowych RGB koloru, a pozostałe 8 bitów umożliwia zapisanie informacji o przezroczystości lub innych dodatkowych właściwościach materiału.

W zależności od potrzeb, dla jednego obiektu lub materiału można wykorzystywać wiele tekstur z różnymi głębiami bitowymi. Rysunek 5.13 przedstawia przykład liścia, którego tekstura koloru opisana jest 24-bitową paletą RGB, a przezroczystość 1-bitową maską alfa, dzięki czemu do realistycznego odwzorowania liścia w przestrzeni 3D wystarczy model złożony z pojedynczego pro-



Rysunek 5.13. Model liścia i jego 24-bitowa tekstura koloru oraz 1-bitowa maska przezroczystości

stokąta z odpowiednio skonfigurowanym materiałem. Warto zwrócić uwagę, że w niektórych przypadkach bardziej opłacalne (ze względu na wydajność aplikacji, zajętość pamięci, liczbę plików itd.) może być stosowanie jednej tekstury 32-bitowej (24 bity na kolor i 8 bitów na dodatkową informację, np. przezroczystość lub intensywność odblasków), zaś w innych sytuacjach korzystniejszą (lub jedyną dostępną technicznie) opcją może być wykorzystanie osobnych tekstur koloru i przezroczystości o odpowiednio mniejszej głębi bitowej.

Różne formaty plików stosowane w grafice komputerowej pozwalają na stosowanie różnych ustawień głębi bitowej i zależnie od możliwości danego silnika grafiki 3D wykorzystanie niektórych formatów może być w nim niemożliwe.

5.8 Formaty plików

Dobór formatu plików dla tekstur zazwyczaj jest narzucony poprzez wymagania danego silnika 3D, zaś w obrębie danej aplikacji często stosuje się tekstury zapisane w różnych formatach, co uzależnione jest od ich funkcji w aplikacji

oraz tym, czy w określonej sytuacji lepiej stosować formaty skompresowane czy nieskompresowane. Niektóre silniki, takie jak Unreal Engine, dokonują automatycznej konwersji nieskompresowanych tekstur do formatów obsługiwanych przez daną technologię.

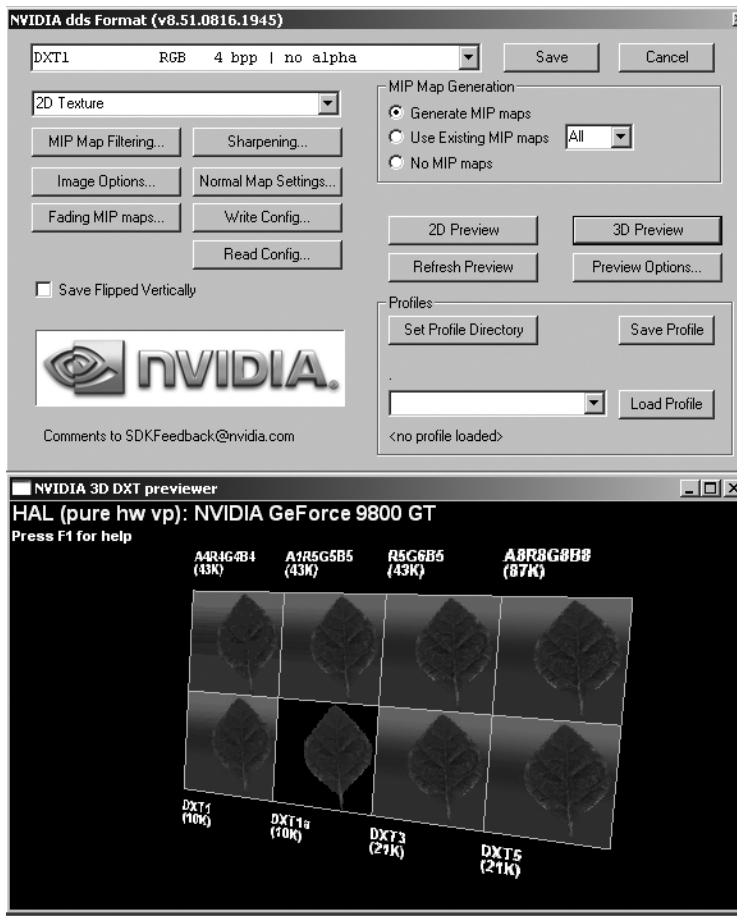
Ze względów historycznych oraz prostoty struktury pliku, jednym z najczęściej stosowanych formatów dla tekstur jest TGA. Jest to format, w którym mogą być zapisywane dane w różnych głębiach bitowych, od 8 do 32 bitów na piksel. Chociaż format TGA może przechowywać dane z kompresją bezstratną, najczęściej tekstury w formacie TGA służą jako pliki źródłowe do późniejszej kompresji stratnej lub też są wykorzystywane w sytuacjach gdzie kompresja stratna byłaby niewskazana. Format TGA jest obsługiwany przez większość silników 3D i programów graficznych.

W przypadku bardziej skomplikowanych tekstur, których utworzenie wymagało stosowania różnych narzędzi i fotografii lub rysunków, pliki źródłowe dobrze jest przechowywać w formacie PSD programu Adobe Photoshop. Format ten pozwala zachować odseparowane warstwy, maski, dodatkowe kanały oraz efekty wykorzystywane przy tworzeniu tekstury, co bywa bardzo przydatne na potrzeby korekcji tekstur na późniejszych etapach pracy nad projektem i generowania nowych tekstur w oparciu o już istniejące. Format PSD standardowo zapisuje dane z kompresją bezstratną. Niestety, zazwyczaj nie można go bezpośrednio wykorzystywać w silnikach 3D i konieczny jest każdorazowy eksport tekstury do jednego z formatów obsługiwanych przez technologię danej aplikacji interaktywnej, jednak z praktycznego punktu widzenia jest to wskazane. Do edycji plików PSD zazwyczaj musimy używać Photoshopa, jednak większość przeglądarek graficznych pozwala przynajmniej na podgląd oraz konwertowanie tych plików do innych formatów.

Dość popularne w licznych zastosowaniach pliki z kompresją stratną JPG i GIF, dla grafiki interaktywnej zazwyczaj nie są zbyt użyteczne. Format JPG pozwala zmniejszyć rozmiar pliku na dysku jednak nie przekłada się to na oszczędność pamięci operacyjnej po wyświetleniu go na ekranie, natomiast format GIF może się sprawdzić tylko w sytuacji gdy potrzebujemy bardzo ograniczonej palety kolorów.

Jednym z najpopularniejszych formatów do kompresji stratnej w przypadku aplikacji interaktywnych jest format DDS. Pozwala on zapisywać 24 lub 32-bitowe pliki tekstur w jednym z kilku dostępnych standardów kompresji. Najczęściej stosowane to DXT1 (24-bitowa tekstura koloru ze współczynnikiem kompresji 6:1) oraz DXT5 (32-bitowa tekstura koloru z kanałem alfa i współczynnikiem kompresji 4:1). Format DDS pogarsza jakość tekstur, szczególnie w przypadku miękkich, gradientowych przejść tonalnych, jednak ze względu na stosunkowo dobre parametry kompresji oraz szybkość dostępu do danych poprzez procesor graficzny jest to często optymalne rozwiązanie. Dodatkowo, format DDS pozwala automatycznie generować mipmapy, czyli dodatkowe tekstury o zmniejszonej rozdzielczości, wykorzystywane przy skalowaniu wyjściowego obrazu w silniku 3D a także zapisywać cubemapy, czyli tekstury złożone z kilku obrazów reprezentujących trójwymiarowe otoczenie sceny. Wiele pro-

gramów do grafiki 2D nie ma wbudowanej obsługi formatu DDS, jednak pakiety 3D, takie jak 3ds Max czy Maya bez problemu ładują, konwertują i zapisują pliki z tym rozszerzeniem. W przypadku Photoshopa można pobrać odpowiedni plugin o nazwie *NVIDIA Texture Tools* ze strony `www.developer.nvidia.com`, który pozwala m.in. otwierać i zapisywać pliki DDS analogicznie do innych formatów obsługiwanych przez program. Podczas zapisu tekstury DDS w Photoshopie możemy wybrać m.in. standard DXT, opcje generowania mipmap i obejrzeć podgląd tekstury w widoku 3D (rysunek 5.14).



Rysunek 5.14. Okno dialogowe do zapisywania tekstur DDS w Photoshopie oraz podgląd tekstury w różnych formatach kompresji

Plików zapisanych z kompresją stratną (np. JPG czy DDS) nie powinniśmy wielokrotnie otwierać i zapisywać w celach edycyjnych, ponieważ przy każdym zapisie może nastąpić pogorszenie jakości obrazu. Gdy planujemy korzystać

z formatu DDS, najlepiej jest przechowywać źródłową teksturę w formacie nieskompresowanym (PSD, TGA itp.) zaś po edycji eksportować ją do formatu skompresowanego.

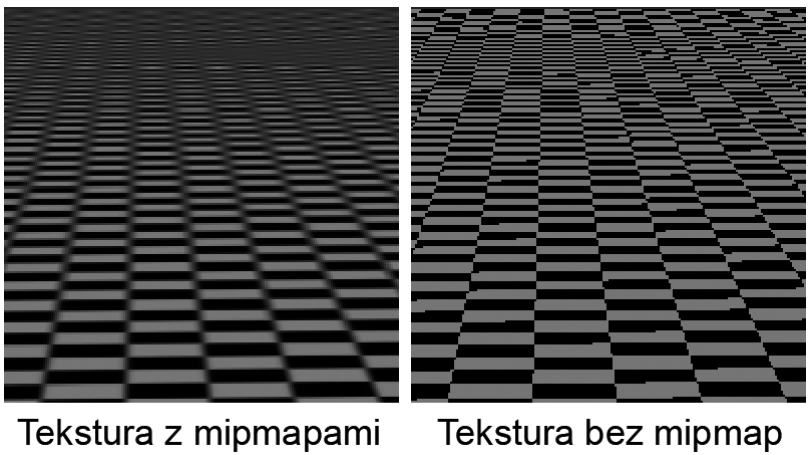
5.9 Mipmapping

Z wykorzystaniem obrazów cyfrowych jako tekstur wiąże się problem dotyczący ich skalowania w zależności od pozycji i orientacji obiektów względem kamery. W przypadku gdy obiekt lub jego fragment pokryty teksturą znajduje się w pewnej odległości od kamery, konieczne jest skalowanie tekseli (jednostek tekstury) w dół, do tego często w sposób nierównomierny związany ze zniekształceniem perspektywicznym. Oznacza to, że moc procesora graficznego zużywana jest na przekształcanie nadmiarowych danych, ponieważ w takim miejscu można byłoby wyświetlić mniejszą teksturę zamiast pomniejszonych tekseli dużej tekstury, a oprócz tego transformowanie tekstur w czasie rzeczywistym często prowadzi do niepożądanych artefaktów. Rozwiązaniem tego problemu jest wykorzystanie techniki mipmappingu, czyli utworzenia serii tekstur o różnych rozmiarach i przełączania ich w zależności od odległości do kamery (rysunek 5.15).

Rysunek 5.15. Tekstura 512x512 pikseli wraz z jej mipmapami

Standardowo każda kolejna mipmapa ma rozmiar o połowę mniejszy od pierwotnej tekstury lub poprzedniej mipmapy i liczba generowanych mipmap dla obrazu o rozmiarze $2^n \times 2^n$ wynosi n (najmniejsza mipmapa ma 1×1 piksel). Oznacza to zwiększenie wymagań pamięciowych o $1/3$ w stosunku do tekstury bez mipmap, jednak zysk na szybkości obliczeń i jakości obrazu jest wart tej ceny. Mipmapping jest obecnie wspierany sprzętowo przez większość kart graficznych i możliwe jest automatyczne generowanie mipmap przy zapisywaniu formatu DDS z Photoshopa.

Podczas wyświetlania tekstur z wykorzystaniem mipmappingu system wybiera automatycznie, która mipmapa jest optymalna dla wyświetlenia tekstury usytuowanej w danej odległości od kamery. Jeśli głowa postaci z rysunku 5.15 na obrazie w rozdzielczości 1280×1024 zajmie więcej niż połowę wysokości ekranu, wtedy wyświetlana jest tekstura główna w rozdzielczości 512×512. Gdy postać znajduje się w takiej odległości od kamery, że jej głowa zajmuje około 35 pikseli, wtedy do jej oteksturowania wystarczy 5. mipmapa (mająca 32×32 piksele). Gdy obraz nie posiada mipmap, program musi przeliczać teksturę o wielokrotnie wyższej rozdzielczości. W praktyce najczęściej dokonywana jest interpolacja pomiędzy różnymi mipmapami z danego przedziału rozdzielczości. Ponieważ mipmapy mogą być wygenerowane poprzez skalowanie tekstury z wielokrotnym próbkowaniem i wygładzaniem krawędzi, więc jakość skalowanego obrazu może być wyższa niż w przypadku transformacji w czasie rzeczywistym i tym samym zmniejszany jest też problem aliasingu (schodkowania krawędzi). Rysunek 5.16. przedstawia porównanie wycinka tej samej powierzchni wyświetlanego z dala od kamery z teksturą posiadająca mipmapy i bez nich.



Rysunek 5.16. Porównanie wyświetlania tekstury z mipmapami i bez mipmap

6

Tworzenie fotorealistycznych tekstur - techniki i narzędzia

W poprzednim rozdziale przedstawiono szereg zagadnień i założeń dotyczących stosowania tekstur w grafice interaktywnej, natomiast niniejszy rozdział poświęcony jest praktycznym przykładom technik i narzędzi służących do tworzenia tekstur.

6.1 Pozyskiwanie tekstur

W tym rozdziale skupimy się na różnych narzędziach związanych z tworzeniem i edycją tekstur, jednak zanim przejdziemy do ich opisu, warto też wspomnieć, skąd można pozyskiwać materiały bazowe do dalszej pracy.

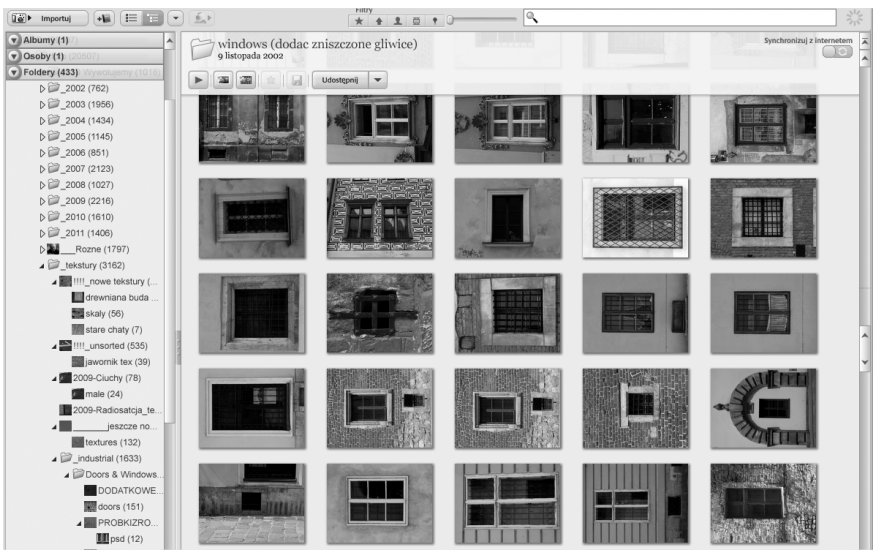
Najpopularniejsze źródła materiałów do tekstur dla grafików 3D w obecnych czasach to:

- zdjęcia wykonywane własnoręcznie aparatami cyfrowymi lub skanowane,
- dedykowane serwisy internetowe (np. CGTextures.com, TurboSquid.com), gdzie za darmo lub za opłatą można pobierać liczne pliki graficzne do wykorzystania w naszych projektach,
- gotowe biblioteki tekstur sprzedawane na płytach CD/DVD (np. przez firmę Dosch Design).

Jeśli tworzony przez nas projekt wymaga stosowania standardowych, realistycznych materiałów, warto sięgnąć do gotowych bibliotek lub serwisów udostępniających interesujące nas materiały, jednak nawet w takich sytuacjach dostosowanie poszczególnych tekstur do wymogów naszych modeli i scen zazwyczaj wymaga od nas sporo pracy. Materiały udostępniane przez innych grafików i firmy często nie są odpowiedniej jakości i prędzej czy później będziemy musieli skorzystać z programów do grafiki 2D, aby uzyskać tekstury o pożądanym wyglądzie i parametrach. Warto też być świadomym faktu, że wykorzystanie w swoich projektach zdjęć pobranych z ogólnodostępnych stron internetowych może rodzić problemy z prawami autorskimi. W przypadku projektów tworzo-

nych na komercyjnych zasadach, zawsze należy też bezwzględnie zadbać o uregulowanie spraw licencji na wykorzystanie danych materiałów źródłowych, co czasami nie jest łatwe, gdyż w różnych krajach obowiązują różne przepisy dotyczące praw autorskich a zasady, na jakich udostępniana jest większość darmowych materiałów w sieci, nie zawsze są jasne.

W trakcie prac nad różnymi projektami warto gromadzić oraz przechowywać wykorzystane do nich materiały źródłowe, tworząc własną bibliotekę tekstur do wykorzystania w kolejnych zadaniach. Niezależnie od ich pochodzenia, zorganizowana w przystępny sposób oraz bogata biblioteka fotografii znacznie przyspiesza prace nad kolejnymi projektami (rysunek 6.1). Do katalogowania zdjęć można wykorzystać wiele różnych darmowych lub płatnych programów, w tym na przykład systemową bibliotekę fotografii Windows czy program Picasa firmy Google.



Rysunek 6.1. Fragment biblioteki tekstur skatalogowanych programem Picasa

6.2 Praca z materiałami fotograficznymi

W trakcie pracy nad teksturami powinniśmy przestrzegać kilku zasad, które pozwalają na efektywne zarządzanie materiałami do projektu, naszym czasem i jakością końcowego efektu. Przy niektórych projektach możemy pozwalać sobie na większą swobodę w podejściu do spraw organizacyjnych i technicznych, jednak z praktycznego punktu widzenia podstawowe metody organizacji pracy nie powinny różnić się niezależnie od skali projektu i jego przeznaczenia.

6.2.1 Dobór materiałów źródłowych

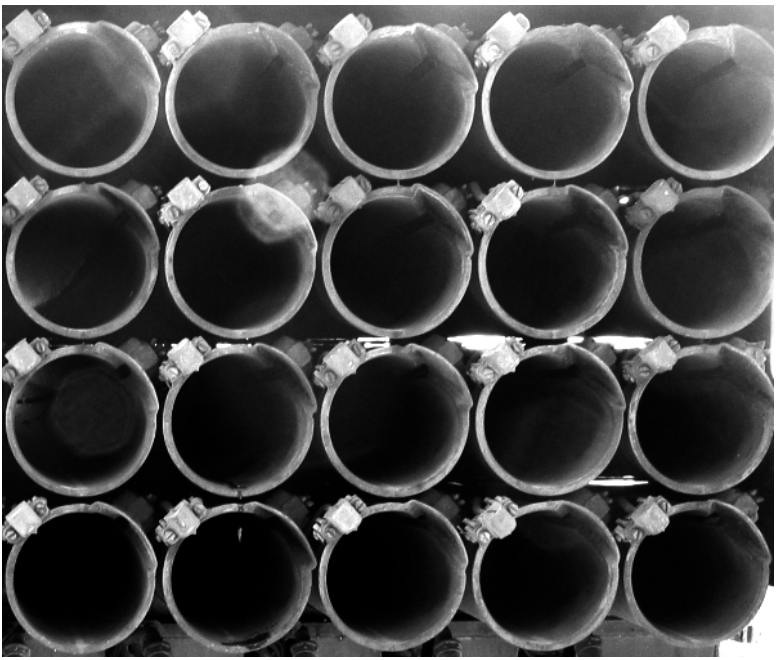
Aby zmaksymalizować szczegółowość i jakość tekstur, najlepiej posługiwać się wyłącznie zdjęciami o odpowiednich parametrach:

- Najlepszym materiałem do teksturowania są zdjęcia o równomiernym rozkładzie oświetlenia, bez widocznych cieni i odblasków (rysunek 6.2).

Rysunek 6.2. Po lewej - przykład zdjęcia źle oświetlonego (cienie i odblaski). Po prawej - zdjęcie oświetlone prawidłowo (równomierny rozkład światła, brak wyraźnych cieni i odblasków)

- Rozdzielczość materiałów źródłowych powinna być większa niż docelowej tekstury.
- Należy unikać zdjęć zawierających zniekształcenia perspektywiczne, beczkowate czy też poduszkowate. W ogromnej większości przypadków tekstu-

Rysunek 6.3. Po lewej - przykład zdjęcia zniekształconego (perspektywa i deformacje beczkowate). Po prawej - zdjęcie jest wykonane prawidłowo (brak perspektywy i łuków)



Rysunek 6.4. Przykład zdjęcia, do którego zakradł się efekt flary i prześwietlenie krawędzi

ra musi przedstawiać płaską powierzchnię bez żadnego efektu głębi obiektu jako całości. Stosunkowo trudno uzyskać jest zdjęcia całkowicie pozbawione tego typu efektów, lecz zdecydowanie powinniśmy minimalizować wkład pracy konieczny w ich korekcję.
- Zdjęcia służące jako tekstury nie powinny być niedoświetlone, prześwietlone ani zbyt nasycone.
- Ostrość zdjęcia powinna być możliwie jak najwyższa, unikajmy zdjęć rozmytych (nawet delikatnie).
- Unikajmy zdjęć zawierających artefakty takie jak winietowanie, aberracje chromatyczne, flary czy duża ilość szumu.

6.3 Wskazówki odnośnie fotografowania na potrzeby tekstur

Gdy do teksturowania obiektów planujemy wykorzystać własnoręcznie wykonane zdjęcia, warto pamiętać o kilku przydatnych wskazówkach:

- W celu uniknięcia zniekształceń perspektywicznych, należy ustawić obiektyw aparatu prostopadle do fotografowanej powierzchni.

- Aby wyeliminować zniekształcenia beczkowate i poduszkowate, konieczne jest odpowiednie dobranie zbliżenia (zoomu) obiektywu.
- Fotografowanie w plenerze najlepiej wykonywać przy rozproszonym oświetleniu słonecznym (dość jasny dzień, ale niebo zachmurzone, brak cieni i odblasków od światła słonecznego).
- Każdą powierzchnię warto sfotografować wiele razy (można przy tym eksperymentować z ustawieniami aparatu), gdyż zazwyczaj na małym ekraniku LCD aparatu cyfrowego nie jesteśmy w stanie sprawdzić czy zdjęcie wyszło odpowiednio ostre.
- Zdjęcia powinniśmy wykonywać w maksymalnej optycznej rozdzielczości aparatu i zapisywać je w nieskompresowanych lub jak najmniej skompresowanych plikach.
- Przy okazji fotografowania interesującej nas powierzchni, warto też wykonać serię zdjęć otoczenia - nigdy bowiem nie wiadomo, jakie elementy mogą przydać się dodatkowo w danej scenie na późniejszym etapie, a oprócz tego dzięki dodatkowym fotografiom łatwiej będzie nam dopracować sposób łączenia danej tekstury z pozostałymi w jej otoczeniu (rysunek 6.5).

Rysunek 6.5. Na potrzeby utworzenia pojedynczej tekstury drewnianej ściany wykonano kilkadziesiąt zdjęć przedstawiających różne ujęcia i fragmenty budynku

- Unikajmy zbyt wysokich ustawień czułości aparatu, które prowadzą do zaszumienia obrazu.
- W zależności od czasu naświetlania optymalnego dla danej sceny, często bardzo przydatnym narzędziem dla twórcy tekstur staje się statyw, nawet najprostszy. W zdjęciach wykonywanych na potrzeby tekstur zawsze chcemy zachować jak największą ostrość, a nawet drobne poruszenia aparatu w trakcie fotografowania mogą nam to utrudnić. Przy czasie naświetlania dłuższym niż 1/80 sekundy zdjęcia bez statywu zazwyczaj wychodzą za mało ostre.
- Jeżeli tylko jest to możliwe, nie stosujmy standardowej lampy błyskowej, w razie potrzeb doświetlajmy scenę źródłami dającymi jak najbardziej rozproszone światło.
- W wielu sytuacjach bardzo pomocny jest odchylany ekranik LCD, z którym można spotkać się w niektórych modelach aparatów cyfrowych. Aby ustawić aparat prostopadle do powierzchni i na odpowiedniej wysokości, często musimy podnosić go nad głowę lub opuszczać nisko nad podłoże, co przy sztywno osadzonym w obudowie aparatu ekranie podglądu uniemożliwia nam prawidłowe ustawienie kierunku i kadru.

Rysunek 6.6. Odchylany ekran podglądu w aparacie cyfrowym bardzo ułatwia kadrowanie zdjęcia

6.4 Zarządzanie plikami

W branżach związanych z grafiką 3D stosuje się różne metodologie i narzędzia do organizowania zasobów projektu i na łamach niniejszej książki nie ma sensu przytaczać opisu żadnego z tych podejść w szczegółach, jednak pewne wytyczne można przyjąć bez względu na środowisko, w jakim pracujemy:

- Pliki z teksturami powinny być już od samego początku projektu trzymane w odpowiednio nazwanych katalogach, nazywane stosownie do ich przeznaczenia. W trakcie prac najczęściej przybywa coraz więcej materiałów i bez określonego nazewnictwa plików nasze zasoby szybko ogarnia chaos. Dobrze przyjętym zwyczajem jest nazywanie plików z teksturami tymi samymi nazwami, jakie mają modele lub sceny, które wykorzystują te pliki.
- Warto upewnić się, czy dana technologia bądź aplikacja zezwala na wykorzystanie polskich liter w nazwach plików oraz czy rozróżnia wielkie i małe litery - wbrew pozorom problem bywa dość istotny, bowiem ścieżki dostępu do plików, ich nazwy oraz automatycznie generowane wersje różnych tekstur mogą na pewnych etapach pracy wymagać zmian w kodzie aplikacji lub przenoszenia repozytoriów plikowych co nawet gdy nie jest skomplikowane technicznie, bywa czasochłonne. Analogicznie jest z długością nazw plików i ścieżek dostępu - niektóre systemy do grafiki interaktywnej mają ograniczenia związane z tymi właściwościami plików.
- Należy określić w jakich formatach chcemy zapisywać skompresowane i nieskompresowane wersje tekstur i do edycji wykorzystywać tylko te drugie. Edycja i zapisywanie uprzednio skompresowanych plików odbija się negatywnie na jakości materiałów docelowych. Niezależnie od schematu pracy, zawsze powinniśmy zachować kopie oryginalnych tekstur w formacie źródłowym.
- Dobrą reguła jest regularne tworzenie kopii zapasowych plików a przy projektach, nad którymi pracuje większa liczba osób, wprowadzenie systemu kontroli wersji plików, który umożliwia odzyskanie utraconych danych i śledzenie lub cofanie zmian w zbiorach danych.

6.5 Rozdzielczość tekstur

Na początku projektu należy przeanalizować technologiczne aspekty związane z rozdzielczością tekstur, przedstawione w rozdziale 5., jednak ze względu na sam proces produkcyjny powinniśmy przestrzegać jeszcze kilku dodatkowych reguł:

- Zawsze należy pracować na materiałach źródłowych w możliwie jak najwyższej rozdzielczości - skalowanie tekstur w dół zazwyczaj nie jest żadnym problemem, skalowanie w górę najczęściej jest bezcelowe lub wręcz szkodliwe dla jakości i wydajności aplikacji. Obecnie w grafice interaktywnej dość rzadko wykorzystuje się pliki o rozmiarach większych niż 1024 i 2048 pikseli, lecz posiadanie dostępu do materiałów źródłowych w większych rozdzielczościach pozwala generować lepszej jakości materiały wynikowe, nawet jeżeli docelowo mają one o wiele niższe rozmiary. W praktyce obróbka zbyt dużych plików może stwarzać problemy z wydajnością komputera, dlatego nie zawsze opłaca się praca na maksymalnej rozdzielczości źródłowej fotografii, jednak nigdy nie należy zmniejszać rozdzielczości obrazu poniżej docelowej rozdzielczości tekstury.

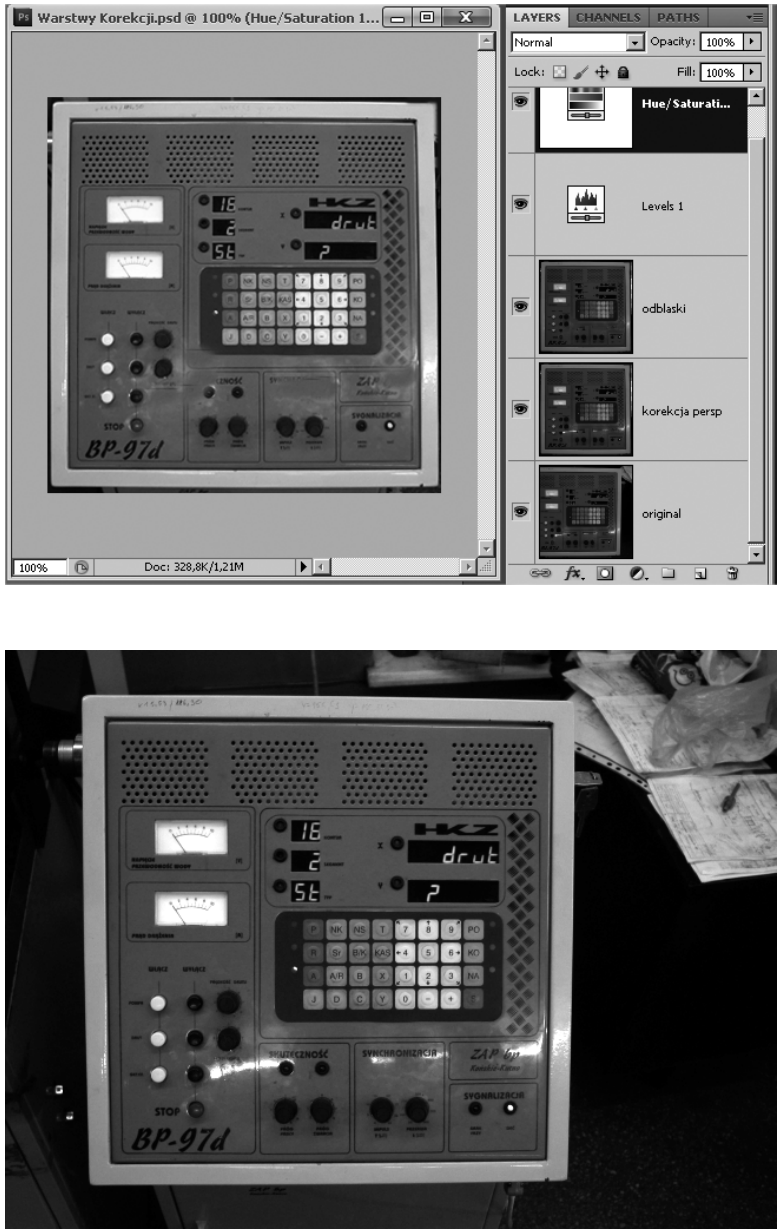
- W grafice interaktywnej rozdzielczości tekstur zazwyczaj powinny być potęgami liczby 2. Jest to związane między innymi z efektywnością alokacji pamięci, zarządzaniem mipmapami oraz standaryzacją wielu narzędzi zarówno po stronie aplikacji graficznych jak też technologii interaktywnych. Dlatego nawet wtedy, gdy dana technologia akceptuje pliki o niestandardowych rozdzielczościach, bezpieczniej jest stosować rozmiary takie jak 256, 512, 1024 czy 2048 pikseli.
- Chociaż większość technologii poprawnie wyświetla tekstury o niemal dowolnych proporcjach, zazwyczaj najkorzystniej jest tworzyć obrazy kwadratowe - niektóre technologie mają problemy z renderingiem tekstur o innych proporcjach, a oprócz tego część narzędzi w różnych programach graficznych została zaprojektowana z myślą o edycji kwadratowych obrazów i praca nad teksturami innymi niż kwadraty może być w nich uciążliwa.

6.6 Operacje edycyjne

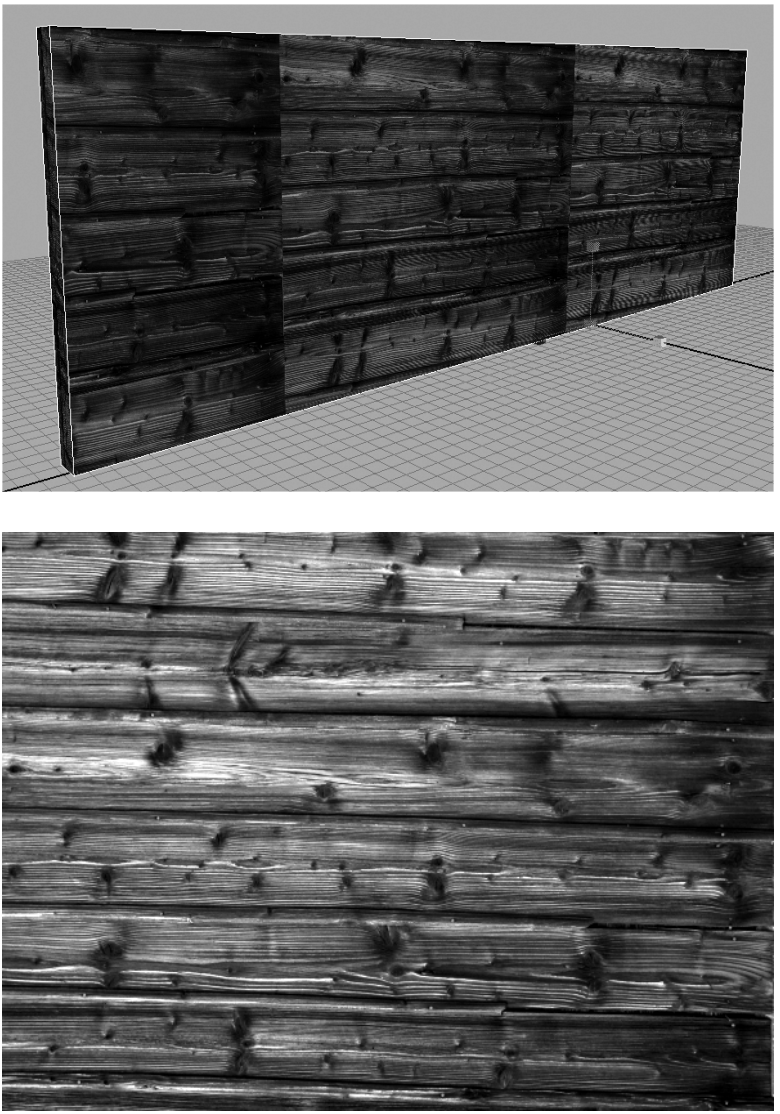
- W przypadku kolażu różnych materiałów fotograficznych w obrębie jednej tekstury, nie powinniśmy skalować w górę fragmentów zdjęć wklejanych do obrazu. W celu zachowania jak najwyższej ostrości tekstury, najlepiej posłużyć się materiałami o rozdzielczości większej niż docelowa i skalować je w dół po wklejeniu do tekstury.
- Dobrze jest unikać operacji powodujących rozmycie obrazu, w tym nie tylko filtrów rozmywających ale także obracania obrazu o kąty inne niż wielokrotność 90°, skalowania oraz różnego rodzaju nieliniowych transformacji kształtu.
- Operacje dotyczące korekcji całego obrazu należy w miarę możliwości wykonywać w sposób umożliwiający cofnięcie zmian - np. poprzez stosowanie łatwych do wyłączenia warstw dopasowania (Adjustment Layers) w Photoshopie lub poprzez zachowanie kopii danej warstwy sprzed oraz po transformacji. Niektóre operacje edycyjne powodują bezpowrotną utratę części danych z obrazu - nie chodzi tylko o ręczne usuwanie fragmentów warstw za pomocą gumki, ale też na przykład korekcję tonalną, w wyniku której eliminujemy pewne poziomy lub kolory z palety obrazu (rysunek 6.7).

6.7 Tworzenie i edycja tekstur w Photoshopie

W kolejnych podrozdziałach przedstawimy przykłady wykorzystania programu Photoshop do obróbki tekstur metodami, jakie często stosuje się przy tworzeniu grafiki interaktywnej. Zamieszczone tu operacje można wykonywać także w innych programach graficznych, a dobór stosunkowo podstawowych narzędzi pozwoli czytelnikowi na realizowanie ich w niemal dowolnej wersji Photoshopa.



Rysunek 6.7. U góry - każdy etap korekcji tekstury w Photoshopie został zapisany na osobnej warstwie (kolejno: korekcja perspektywy, usuwanie odblasków, korekta tonalna i redukcja nasycenia). U dołu - oryginalne zdjęcie wykorzystane w tej teksturze



Rysunek 6.8. Model drewnianej ściany utworzony z prostopadłościanu i fotografia wykorzystana jako jego tekstura

6.7.1 Korekcja fotografii źródłowej

Na rysunku 6.8 przedstawiono prostą bryłę 3D, która dzięki odpowiedniej powtarzającej się (zapętlonej) teksturze ma stanowić model drewnianej ściany. Źródłowa fotografia tekstury pokazana jest w dolnej części rysunku. Już pat-

rząc na samą fotografię widać, że tekstura nie będzie prawidłowo powtarzana na powierzchni obiektu, ponieważ linie poszczególnych desek zostały sfotografowane pod pewnym kątem i biegną na ukos. Nie zawsze możliwe jest wykonanie fotografii pod odpowiednim kątem, nie zawsze też w trakcie fotografowania jesteśmy w stanie potwierdzić poprawność perspektywy, stąd nawet pomimo przestrzegania zasad wymienionych we wcześniejszej części rozdziału może się okazać, że konieczna jest korekcja geometrii zdjęcia już podczas jego obróbki w programie graficznym.

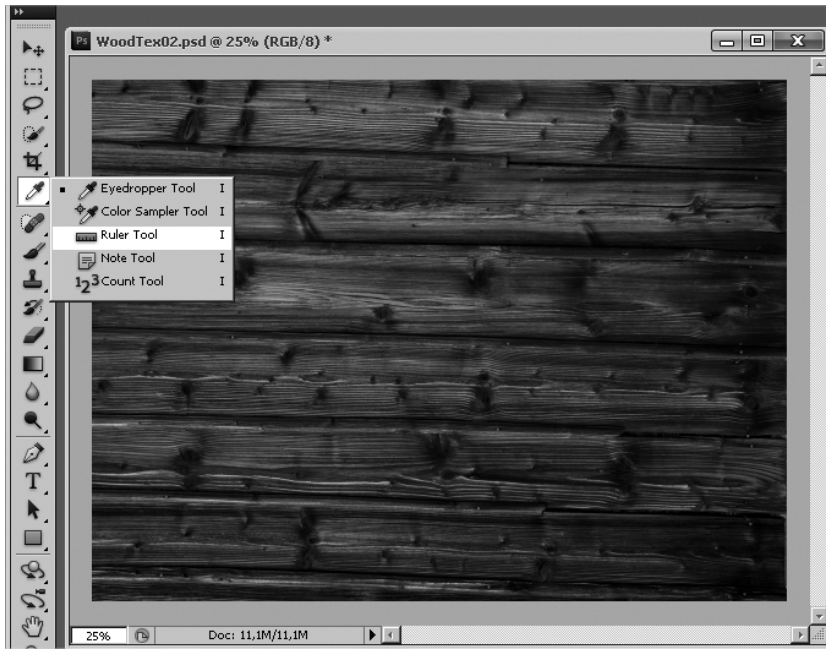
Dodatkowym problemem jest przejście tonalne biegnące w poprzek zdjęcia - po prawej stronie deski są wyraźnie ciemniejsze niż po lewej, co na otekstorowanym obiekcie daje efekt cienia pojawiającego się przy każdym powtórzeniu tekstury i podkreślającego krawędź zdjęcia. Tego typu efekt może się wiązać bądź z nierównomiernym oświetleniem fotografowanej sceny, bądź z naturalnymi właściwościami danej powierzchni. Jeśli tekstura nie musi być zapętlona i powtarzana na powierzchni obiektu, można go niekiedy pominąć, jednakże w grafice interaktywnej najczęściej stosujemy tekstury powtarzane w różnych konfiguracjach i konieczna jest eliminacja tego typu przebarwień lub zmian luminancji.

W omawianym przypadku, by ściana nabrała odpowiedniego wyglądu musimy skorygować zarówno efekt perspektywy jak i tonację zdjęcia.

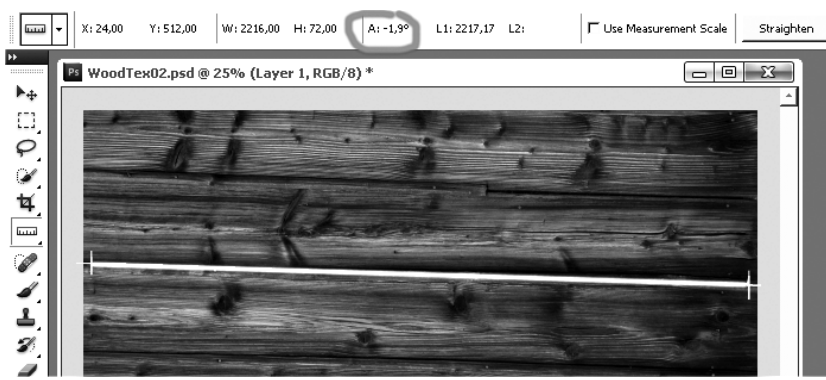
6.7.2 Korekcja perspektywy i kadrowanie

Po załadowaniu zdjęcia do Photoshopa należy wybrać narzędzie linijki (Ruler Tool) z przybornika po lewej stronie ekranu (rysunek 6.9), a następnie kliknąć na początku jednej ze szczelin między deskami i przeciągnąć kursorem w bok, tak aby rozpiąć linijkę na całej jej długości (rysunek 6.10). Na górnej listwie informacyjnej programu wyświetlona zostanie informacja o rozmiarze i kącie nachylenia linijki (w naszym przypadku około 2^0), który odpowiada wartości, o jaką musimy skorygować deski na teksturze.

Po ustawieniu linijki należy wybrać polecenie *Image/Image Rotation/Arbitrary* z górnej listwy menu - w okienku Rotate Canvas powinna automatycznie wyświetlać się wartość obrotu odczytana z linijki. Po kliknięciu przycisku *OK* obrazek zostanie obrócony o odpowiedni kąt i ukośne liniie desek zostaną wypoziomowane. Niestety, powtarzalność tekstury pogorszy się w wyniku tej operacji jeszcze bardziej, ponieważ krawędzie obrazka stały się teraz ukośne i przebija zza nich białe lub czarne tło (zob. rysunek 6.11). Jest to dobra chwila aby skadrować teksturę do odpowiednich rozmiarów i proporcji. Na rysunku 6.11 pokazano kwadratową ramkę zaznaczenia, do której obraz zostanie skadrowany (aby utworzyć taką ramkę, należy włączyć narzędzie prostokątnego zaznaczania (klawisz M) i przytrzymać klawisz Shift podczas rozciągania ramki. Ramka została rozciągnięta od najwyższej krawędzi, przy której można rozpiąć kwadrat o maksymalnej dla tego obrazka powierzchni, do najniższej krawędzi spełniającej ten sam warunek.

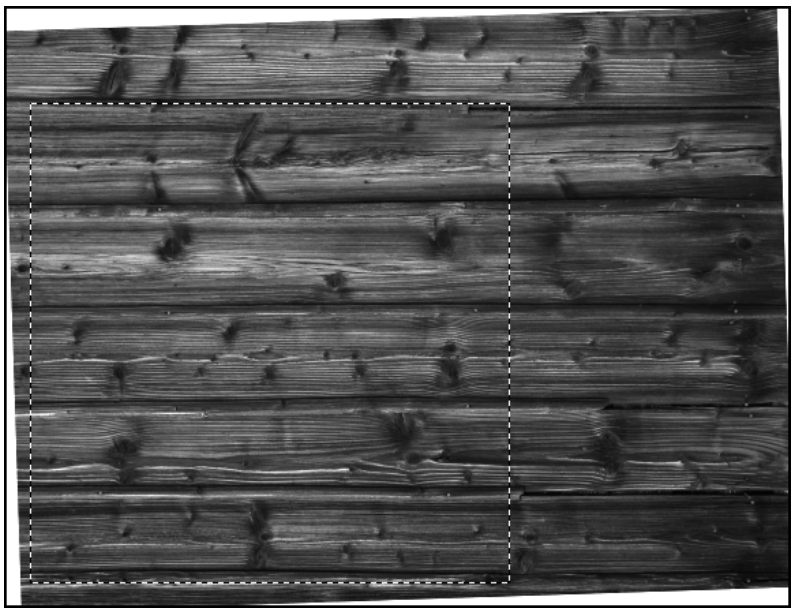


Rysunek 6.9. Wybór narzędzia Ruler Tool w Photoshopie



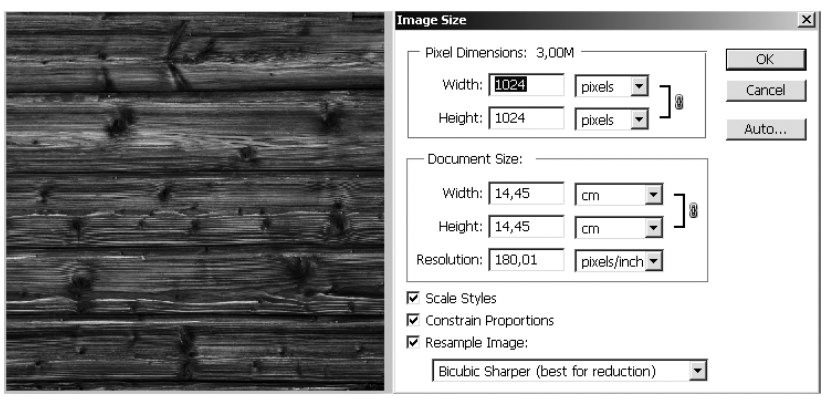
Rysunek 6.10. Rozciągnięta linijka i informacje o kącie jej nachylenia

Ze względów wspomnianych wcześniej w tym rozdziale, tekstura o proporcjach kwadratu jest optymalnym rozwiązaniem dla grafiki interaktywnej, jednakże nam zależy na wykorzystaniu maksymalnie dużej powierzchni zdjęcia pierwotnego, żeby zapętlony fragment zdjęcia miał stosunkowo dużą rozdzielczość i nie powtarzał się zbyt wiele razy na tej samej ściance obiektu. Skadrowanie fotografii pomiędzy krawędziami dwóch desek ułatwi zapętlenie tekstury.



Rysunek 6.11. Po obróceniu obrazka konieczne jest wyeliminowanie pustego tła

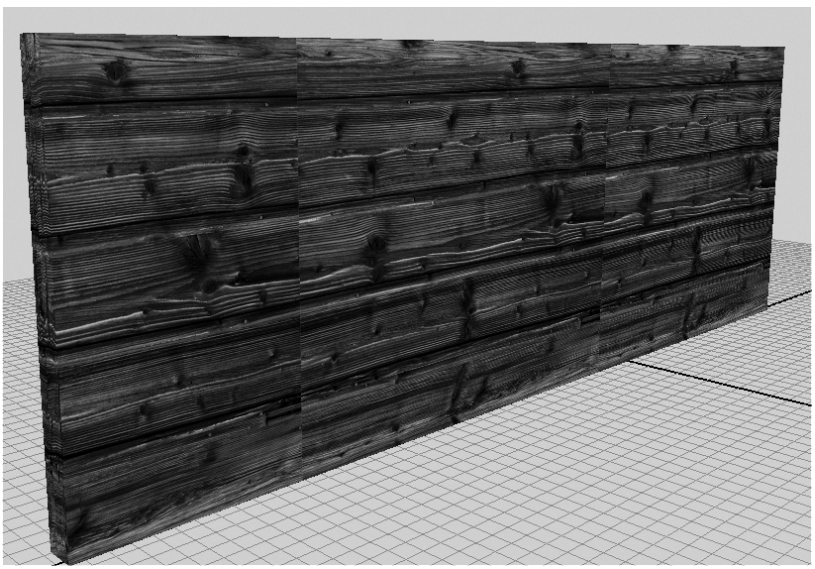
Po rozciągnięciu kwadratowej ramki zaznaczenia wybieramy polecenie *Image/Crop*, w wyniku którego obraz stanie się kwadratem. Poleceniem *Image/Image Size* sprawdzamy aktualny rozmiar obrazka (rysunek 6.12). W naszym przypadku możemy przyjąć (dość arbitralnie), że wystarczającą rozdzielczością tekstury, zapewniającą dobrą jakość renderingu i odpowiednią wydajność, jest 1024×1024 pikseli i takie wartości wprowadzamy w polach Width oraz Hei- ght



Rysunek 6.12. Zmiana rozmiaru skadrowanej tekstury

(o ile rozmiar pierwotnej tekstury nie był mniejszy, należy również pamiętać o unikaniu skalowania w górę). W dole okna Image Size możemy zmienić metodę próbkowania pikseli wykorzystywaną przy skalowaniu. W przypadku tekstur zależy nam na maksymalnej ostrości obrazu, dlatego zmieniamy opcję Resample Image z domyślnej na Bicubic Sharper, która zapewnia lepsze wyostrzenie drobnych szczegółów przy zmniejszaniu obrazka.

Po zapisaniu i przeładowaniu tekstury w programie graficznym lub silniku gier widać, że drewniana ściana wygląda zdecydowanie lepiej, jednak ciągle bardzo jej powierzchnia posiada wyraźne szwy związane ze stykaniem się lewej i prawej krawędzi tekstury przy powtórzeniu obrazka (rysunek 6.13).



Rysunek 6.13. Tekstura po korekcji perspektywy i kadrowaniu

6.7.3 Zapętlanie tekstury

Aby powtarzająca się tekstura na powierzchni obiektu wyświetlana była bez widocznych szwów, jej krawędzie muszą zawierać identyczne lub pasujące do siebie piksele. W przypadku fotografii ten warunek niemal nigdy nie jest spełniony bez przeprowadzenia odpowiedniej korekcji w programie graficznym.

Krawędzie tekstury można dopasowywać do siebie różnymi technikami, jednak najbardziej uniwersalną z nich jest pokrycie ich tymi samymi fragmentami modyfikowanego obrazka. Operacja ta jest dość niewygodna w przypadku obrazka w oryginalnej postaci, lecz przy pomocy jednego z filtrów Photoshopa zadanie to staje się stosunkowo proste.

Po załadowaniu obrazka do Photoshopa i zaznaczeniu odpowiedniej warstwy należy wybrać polecenie *Filter/Other/Offset* i w polach Horizontal i Vertical wpisać wartości będące połową rozdzielczości tekstury (w naszym przypadku 512) - tym samym piksele będące przy krawędziach tekstury znajdą się pośrodku obrazka a piksele ze środka znajdą się przy krawędziach. W tej chwili wystarczy dowolną techniką wygładzić ostre szwy pomiędzy ćwiartkami tekstury i uzyskamy idealnie zapętlony obraz. Dość standardową metodą jest wykorzystanie stempla do klonowania (Clone Stamp Tool - klawisz S), który pozwala kopiować fragmenty tekstury z dowolnego miejsca obrazka i miękko zamalowywać nimi ostrą krawędź. Aby pobrać fragment tekstury należy kliknąć w dowolnym jej miejscu z klawiszem Alt, a następnie malując kursorem po szwie maskować jego obecność. Dla uzyskania większej precyzji, operację tę warto wykonywać w zbliżeniu na dany fragment obrazka (rysunek 6.14).

Rysunek 6.14. Maskowanie szwu na teksturze narzędziem Clone Stamp Tool

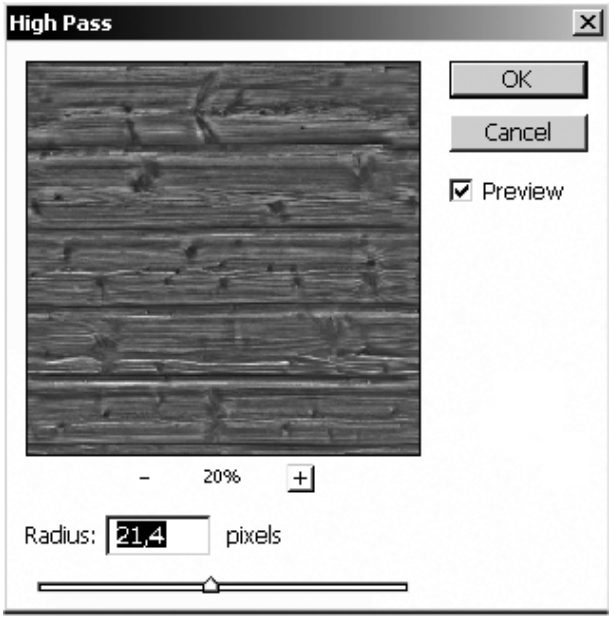
Po zamaskowaniu szwu możemy ponownie użyć polecenia *Filter/Other/Offset*, aby piksele ze środka i z krawędzi obrazka wróciły na swoje pierwotne miejsca. Tak przetworzona tekstura na modelu nie będzie już miała widocznych szwów, jednak ciągle nie rozwiązany pozostaje problem przejścia tonalnego, które tworzy ciemny cień po prawej stronie tekstury (po skadrowaniu obrazka jest on mniej widoczny ale przy większej liczbie powtórzeń tekstury wciąż może niekorzystnie uwydatniać jej zapętlony charakter - zobacz rysunek 6.15).

Aby wyeliminować przyciemnienie, które z matematycznego punktu widzenia ma charakter zakłócenia o niskiej częstotliwości, należy przetworzyć obrazek filtrem górnoprzepustowym, który pozostawi w obrazie tylko składowe o wysokiej częstotliwości, czyli drobne szczegóły tekstury. W tym celu w Pho-

Rysunek 6.15. U góry: przy większej liczbie powtórzeń tekstury uwydatnia się jej przyciemnienie po prawej stronie. U dołu: zapętlona tekstura, wyglądająca dość poprawnie, jednak oko obserwatora łatwo wychwyci cykliczne przyciemnienie i ciemną plamę w jego centrum

toshopie należy wybrać polecenie *Filter/Other/High Pass* i dobrać odpowiednią wartość parametru Radius, w okienku podglądu obserwując zmiany obrazu (rysunek 6.16). Im mniejsza wartość tego parametru, tym więcej przejść tonalnych zostanie odfiltrowanych z obrazu i będzie się on stawał coraz bardziej płaski. Ponieważ ta operacja bezpowrotnie usuwa pewne składniki obrazu, warto wykonywać ją na kopii oryginalnej warstwy, aby w razie potrzeby móc wrócić do wcześniejszej postaci tekstury i skorygować ją ponownie.

Filtr High Pass powoduje utratę kontrastu oraz nasycenia kolorów, dlatego po jego użyciu konieczne jest skorygowanie tych właściwości. By powonie przywrócić kolory obrazu, bezpośrednio po użyciu filtra wybierz polecenie *Edit/Fade High Pass* i z rozwijanej listy Mode wybierz pozycję Luminosity. Oznacza to, że program ograniczy działanie filtra tylko do luminancji obrazu, starając się zachować jego pierwotną kolorystykę. Następnie dodajmy warstwę dopaso-



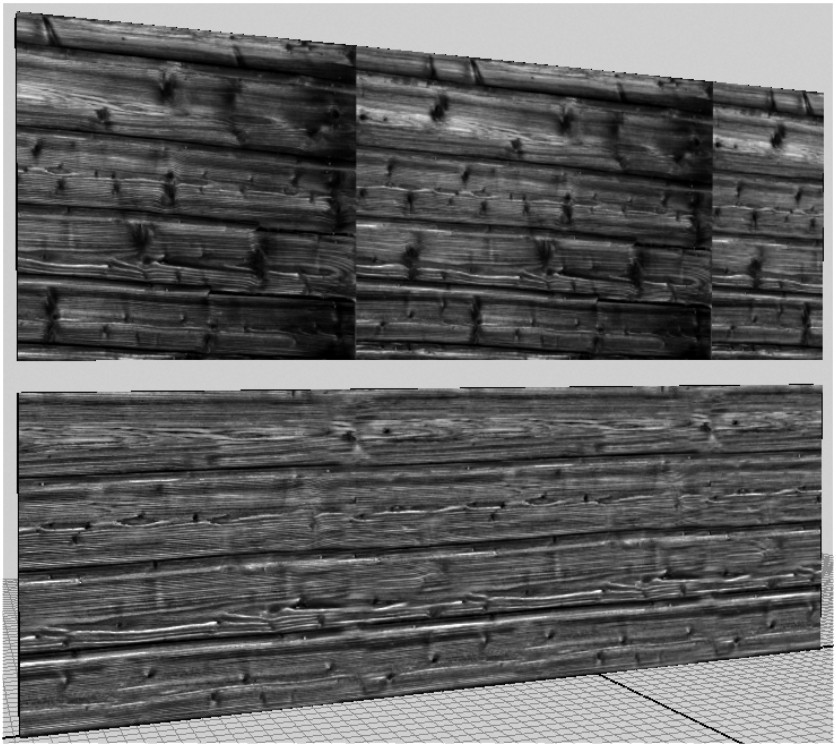
Rysunek 6.16. Wykorzystanie filtru górnoprzepustowego do usunięcia głębokich przejść tonalnych z obrazu

wania Levels lub użyjmy polecenia *Image/Adjustment/Levels* (bądź też innej funkcji do korekty tonalnej) aby nadać obrazowi odpowiedni kontrast.

W razie potrzeby można też skorygować nasycenie kolorów funkcją Hue/ Saturation (Ctrl+U). Po wykonaniu tych operacji obraz powinien mieć kolorystykę i wizualny kontrast zbliżony do oryginalnej fotografii i jednocześnie być pozbawiony głębokich przejść tonalnych. Warto też na tym etapie przeanalizować ogólny pożądany kontrast i nasycenie tekstury i jeśli to konieczne, wspomnianymi wyżej narzędziami przeprowadzić korekcję jasności, kontrastu i kolorów pod kątem tonacji i oświetlenia docelowej sceny.

Na powtarzającej się teksturze zbyt mocno widoczne były liczne i grube sęki, tworzące charakterystyczny wzór, dlatego możemy ponownie wykorzystać narzędzie stempla (Clone Stamp Tool) do zamalowania części z nich. Na rysunku 6.17 przedstawia porównanie modelu ściany z teksturą sprzed korekcji po wykonaniu w Photoshopie wszystkich operacji opisanych powyżej.

Należy zwrócić uwagę na to, że staranna korekcja tekstury wymaga zazwyczaj duzo większej liczby operacji i wielokrotnego ich powtarzania - przykład przedstawiony wyżej jest bardzo prosty koncentruje się na przystępnym opisie podstawowych narzędzi a nie na żmudnej i czasochłonnej pracy, jaką jest zazwyczaj dopracowywanie zawartości tekstury.



Rysunek 6.17. U góry: oryginalna tekstura bez korekcji. U dołu: efekt obróbki tekstury w Photoshopie

7

Oświetlenie w grafice interaktywnej

7.1 Wstęp

Jednym z najważniejszych elementów w grafice 3D jest jej oświetlenie. Podobnie jak w sztuce filmowej czy malarstwie, z którymi grafika multimedialna ma wiele wspólnego, światło może - i przeważnie kreuje - nastrój, podkreśla elementy geometrii sceny. Filmowcy znająsdoskonale pojęcie „magicznej godziny", czyli czasu, w którym naturalne światło słońca staje się miękkie i plastyczne, o odpowiednim zabarwieniu, pozwalającym na ujęcia o głębszym wyrazie emocjonalnym; malarstwo wartość światłocienia jako środka przekazu estetycznego odkryło dawno temu. Pomijajšc szczegóły, warto nadmienić, że najwięksi malarze, tacy jak Rembrandt czy Vermeer, posługiwali się takimi metodami, które dziś stanowią jeden z podstawowych elementów profesjonalnego oświetlenia sceny 3D.

Unreal Development Kit firmy *Epic*, znana zapewne każdemu graczowi, to darmowa wersja jednego z najbardziej zaawansowanych silników graficznych, zwanego *Unreal* i rozwijanego od ponad dekady. Jest to zestaw narzędzi, który oferuje nie tylko renderer graficzny (a więc system „rysowania" grafiki na ekranie) lecz także system fizyki ciał stałych jak też system tworzenia skryptów (zwany *Kismet*) czy system zarządzania wszystkimi elementami tworzonej sceny 3D (tak zwany *Content Browser*). Jest to kompleksowe narzędzie, które swoimi możliwościami oraz łatwością obsługi przebija większość konkurencyjnych rozwiązań.

W powszechnym przekonaniu twórców i graczy (osób najczęściej podziwiających efekty pracy w silniku *Unreal*), silnik ten najlepiej spisuje się w przypadku scen typu *indoor* (wewnątrz pomieszczeń), gorzej zaś w renderowaniu scen typu *outdoor* („na zewnątrz" pomieszczeń). Dzieje się tak, ponieważ *Unreal*, którego pochudną jest *UDK*, został początkowo zaprojektowany tylko do tworzenia scen wewnątrz pomieszczeń, co było podyktowane ograniczeniami technologicznymi końca lat dziewięćdziesiątych. Od wielu lat *Epic* systematycznie rozwija renderer tak, by mógł sprostać scenom typu *outdoor* i obecnie niewiele

lub zgoła w ogóle nie ustępuje on pod względem jakości renderowanego obrazu konkurencyjnym rozwiązaniom.

7.2 Podstawowe właściwości świateł

7.2.1 Sposób działania renderera

Na wstępie należałoby wspomnieć o sposobie, w jaki działa renderer, czyli program odpowiedzialny za wyświetlenie grafiki 3D i 2d na ekranie czy dowolnym urządzeniu wyjściowym i bedący również podstawowym komponentem *UDK*. Komponując oświetlenie dla sceny, należy uwzględnić sposób pracy renderera.

Przede wszystkim, renderer póki ten nie jest oświetlony nie wyświetla (renderuje) żadnego elementu sceny. Oznacza to, że nawet najbardziej skomplikowana i złożona scena nie zostanie wyświetlona na urządzeniu wyjściowym (ekranie, rzutniku, telewizorze, itp) jeżeli nie zostanie dodane do niej światło lub renderer zostanie, odpowiednim poleceniem, przestawiony w tryb jednolitego oświetlania sceny.

Z drugiej strony, renderer wyświetla scenę w specyficzny sposób - jedna klatka obrazu potrafi być renderowana nawet kilkanaście razy nim zostanie przekazana na urządzenie wyjściowe. Po pierwszym podstawowym „przejściu" renderera (ang. *1 pass*, pierwsze przejście), czyli wyrenderowaniu pełnej klatki obrazu, możliwe są następne przejścia, które pozwalają na wprowadzenie tzw. postprocesów (ang. *postprocess*) czyli przetwarzania klatki danego obrazu według zadanego algorytmu. Dzięki temu scena 3D może uzyskać głębię ostrości, mgłę, otoczkę wokół świateł o dużej energii itd.

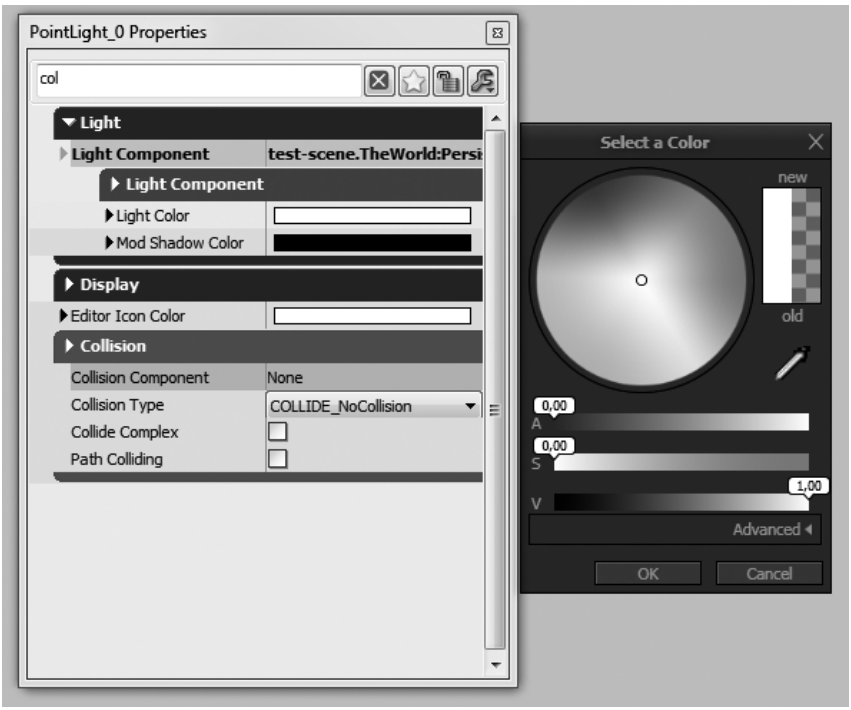
Efekty postprocesowe muszą zostać uwzględnione przy komponowaniu oświetlenia zarówno pod względem kolorystycznym jak i jakościowym, co oznacza, że niektóre postprocesy mogą okazać się zbędne, inne należy użyć z umiarem lub w ograniczonym zakresie. Światło sceny musi „współgrać" z postprocesem pod każdym względem i nie może być traktowane jako dodatek do sceny; podobnie postproces nie może być projektowany jako osobny element.

Aby uzyskać złudzenie rzeczywistości, wszystkie współczesne renderery pozwalają na zastosowanie wielu typów świateł o mnogich, modyfikowalnych parametrach i cechach. Wszystkie światła, nie tylko te, stosowane w *Unreal Development Kit* lecz także w innych programach, mają pewne elementy wspólne, które podyktowane są fizyczną naturą światła i starają się ją wiernie oddać. Od określenia tych cech należy zacząć projektowanie oświetlenia.

7.2.2 Podstawowe cechy świateł - Color

Podstawowym elementem jest kolor światła. Każde światło może mieć przypisany dowolny kolor z dostępnej palety barw i - w większości przypadków - można posłużyć się wartościami RGB (ang. *Red, Green, Blue* - czerwony, zielony, niebieski czyli podstawowe składowe koloru) lub też wybrać bezpośrednio

z wyświetlanej palety. Aby tego dokonać, należy zaznaczyć lewym klawiszem myszy światło, umieszczone w scenie a następnie nacisnąć klawisz F4 by otworzyć okno właściwości światła (*Properties*). Najszybszą metodą jest wpisanie w oknie wyszukiwania fragmentu nazwy opcji (w przypadku *Color* jest to *col*) co spowoduje przefiltrowanie opcji i wyświetlenie tylko tych, które zawierają wpisaną frazę. Oczywiście, można również kliknąć prawym przyciskiem myszy i wybrać *Properties*, następnie odszukać w sekcji *Light* podsekcję *Light Component* a w niej *Light Color*. Obie metody pozwolą na otwarcie okna wyboru koloru (rysunek 7.1).



Rysunek 7.1. Właściwości światła - wybór koloru

7.2.3 Podstawowe cechy świateł - Brightness

Następnym ważnym elementem światła jest jego natężenie (ang. *brightness*), które odpowiada za to, z jaką intensywnością oświetlone są elementy sceny. Ponieważ źródło światła jest niewidoczne, więc natężenie światła można potraktować jak pożądaną „jasność" obiektów, otaczających światło. Odpowiednio dobrane natężenie światła może pozwolić podkreślić cechy geometryczne elementów sceny, dodając szczegółów, lecz zbyt duże natężenie może powierzch-

nie trójwymiarowych obiektów dosłownie „wyprać" ze szczegółów, zostawiając zbyt jasne, prześwietlone powierzchnie. *Brightness* wybiera się podobnie jak opcję *Color*, z tą różnicą, iż, kliknąwszy dwa razy na polu obok tekstu *Brightness*, podaje się wartości liczbowe, gdzie 0 jest brakiem światła a wartości większe niż 20 powodują często prześwietlenie sceny.

Rysunek 7.2. Właściwości światła - *Brightness*

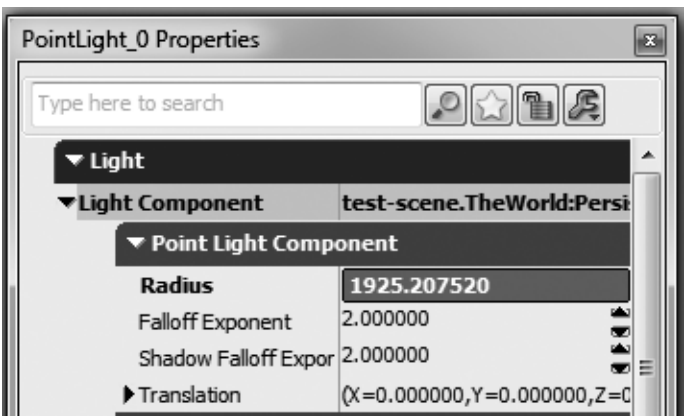
7.2.4 Podstawowe cechy świateł - Radius

Ważnym elementem światła jest jego zasięg (ang. *radius*) zawsze rozumiany jako sfera, poza którą światło nie dochodzi. Oczywiście współczesne programy

renderujące pozwalają na wykluczenie z oświetlania pewnej strefy sceny 3D, znajdującej się w zasięgu światła (ang. *exclusion volumes*). Zasięg światła jest tak pomyślany, że punkt w przestrzeni 3D, w którym umieścimy światło, jest środkiem sfery światła, natomiast punkty na obrzeży stanowią miejsce w scenie 3D, gdzie natężenie światła spada do zera. Pozwala to na oświetlanie sceny zgodnie z fizycznym modelem światła, przynajmniej w pewnym uproszczeniu. Edycji wartości *Radius* dokonuje się podobnie jak wartości *Brightness*.

7.2.5 Podstawowe cechy świateł - Fallof

Cechą, współgrającą z natężeniem światła, jest jego rozpraszanie. *Falloff Exponent*, odpowiedzialny za rozpraszanie światła, powoduje, że światło, od pewnego określonego punktu w przestrzeni, rozchodzi się wg innego algorytmu, przez co krawędzie oświetlonej sfery, wyznaczanej przez zasięg światła, stają się bardziej lub mniej rozmyte; światło rozprasza się łagodniej, bardziej miękko lub wręcz przeciwnie, staje się twarde i ostre.



Rysunek 7.3. Właściwości światła - *Radius* i *Fallof*

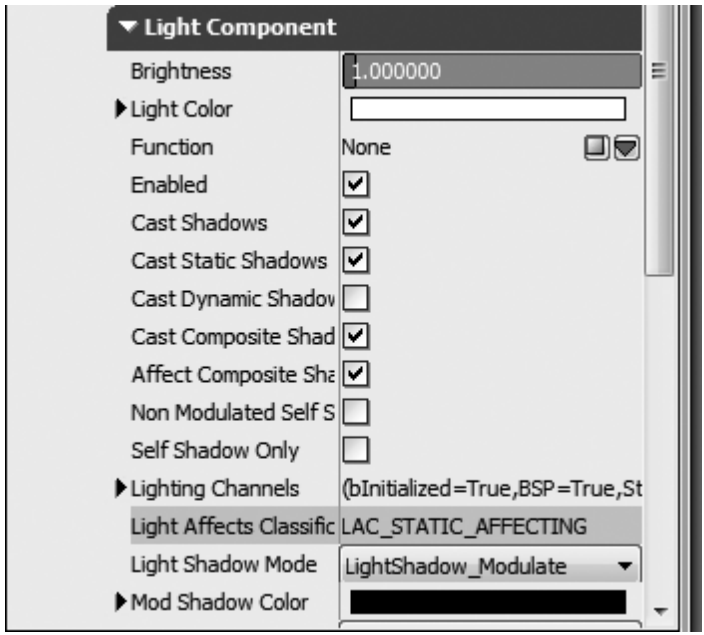
7.2.6 Podstawowe cechy świateł - Shadows

Cienie (ang. *shadows*), są jednym z tych elementów oświetlenia sceny, które nadają jej pozory realizmu. Niestety, ich generowanie może stanowić również jedno z najbardziej obciążających renderer zadań - zwłaszcza, gdy mowa o renderowaniu cieni dynamicznych, tzn. rzutowanych przez zmieniające się, ruchome elementy sceny.

W *UDK* stosowane są tzw. *lightmaps* (zwane mapami oświetlenia lub mapy propagacji świateł), przechowujące informacje o rozkładzie świateł oraz cieni

a) Scena z cieniami

Rysunek 7.4. b) Scena bez cieni



Rysunek 7.5. Właściwości światła - *Shadows*

w scenie. *Lightmaps* są generowane tylko dla obiektów statycznych. Jeśli scena zawiera tylko takie obiekty, możliwe jest uzyskanie niskim nakładem pracy realistycznych efektów. Należy jednak pamiętać, że, o ile w teorii każde światło może rzucać cienie, o tyle w praktyce takie podejście, choć mogłoby się wydawać, dające najlepsze efekty, jest całkowicie błędne. Po pierwsze, niepotrzebnie obciąża renderer w procesie renderowania sceny, zwłaszcza dynamicznej, a po drugie, może (i najczęściej też powoduje) błędy w procesie renderowania cieni, takie jak rozmycie ich krawędzi, prowadzące nawet do całkowitego ich zaniku. W *UDK* istnieje kilka sposobów renderowania cieni. *Normal shadows* pozwalają renderować cienie, które wzajemnie się „znoszą", osłabiają; *modulated shadows*, kosztem większego obciążenia renderera pozwalają na stworzenie wielu cieni jednego obiektu, rzutowanych z wielu źródeł światła. Podobnie *Modulate_better*, które podnoszą jakość cieni renderowanych za pomocą tego samego algorytmu co w przypadku *modulated shadows*.

Dobrym nawykiem w komponowaniu oświetlenia dla sceny 3D jest dokładne zaplanowanie, które światła rzucają cienie. Dodatkowo, należy zawsze pamiętać o celowości użycia cieni i oszczędności, świadczącej o zrozumieniu procesu renderowania cieni.

Poprzez zaznaczenie opcji *Cast shadows* we właściwościach światłą, światło będzie rzucać cienie.

7.3 Zasady komponowania oświetlenia sceny 3D z wykorzystaniem cieni

Scena oświetlona i scena wykorzystująca cienie to dwa, zupełnie inne pojęcia, tworzące też zupełnie inną jakość. W pierwszym wypadku efekt będzie mniej realistyczny, choć, w przypadku scen o mniejszym stopniu skomplikowania, różnica może wydawać się niezauważalna, w drugim nawet scena, która w zamyśle nie miała odzwierciedlać znanej rzeczywistości, może wydawać się - w jakimś stopniu - mimetyczna. Przykładowo, scena, przedstawiająca krajobraz obcej planety nie będzie realistyczna w dosłownym tego słowa znaczeniu, nie będzie przedstawiać rzeczywistości znanej odbiorcy, jednak, dzięki odpowiednio zaplanowanemu oświetleniu i planowo użytym cieniom, może stać się mimetyczna, tzn. naśladować cechy znanego nam świata.

Cienie mogą być, jak już wspomniano, rzucane przez każde światło w scenie; jednak, zanim powstanie oświetlenie sceny, należy wybrać te światła, które sprawią, że rzutowany cień będzie nie tylko miał sens (istnienie rzucającego go światła będzie uzasadnione logicznie), lecz będzie też stanowił artystyczną wartość naddaną - wprowadzi do sceny jakiś dodatkowy element budujący, na przykład nastrój.

Jeżeli jakiś fragment sceny oświetlany jest kilkoma światłami, jedno z nich może rzutować cień. Jednakże, w wielu wypadkach, im więcej świateł tym cień będzie bardziej rozmyty. Dlatego też jedną z ciekawszych metod na oświetlenie oraz „ocienienie" sceny typu *outdoor* jest wykorzystywanie do rzucania cieni silnego światła typu *Directional Light* (ang. światła kierunkowego), symulującego światło dzienne. W ten sposób światło dominujące rzutuje wszystkie cienie, natomiast inne elementy sceny są dopełnione światłami pozbawionymi cieni.

Podobnie w przypadku scen wewnątrz pomieszczeń (tzw. *indoor*), cień powinien być rzutowany przez silne, dominujące światła, nie rozpraszane wpływem innych świateł. Dodatkowo w obydwu przypadkach należy zwrócić uwagę na powierzchnię, na jaką będzie rzutowany cień. Zbyt ciemna, o niewłaściwie skontrastowanym materiale, może spowodować, że cień będzie rozmyty albo zgoła niewidoczny. W przypadku *UDK* sytuacje może jeszcze poprawić zwiększenie rozdzielczości *lightmap* (map, zawierających rozkład świateł i cieni na każdym elemencie sceny) lub, w przypadku korzystania z zaawansowanych technik renderingu jak *Lightmass*, zmiana parametrów globalnego oświetlenia sceny takich jak *Diffuse*.

Podsumowując, rzutowanie cieni przez światło jest jedną z najciekawszych i dających największe możliwości technik; by w pełni wykorzystać potencjał, tkwiący w renderowaniu cieni, należy używać ich z umiarem i tylko tam, gdzie ich istnienie jest uzasadnione logicznie i stanowi artystyczną wartość naddaną. Trzeba też brać pod uwagę obciążenie procesora podczas renderowania scen, co jest szczególnie ważne w przypadku scen renderowanych w czasie rzeczywistym, takich jak gry wideo czy inne aplikacje interaktywne.

Oczywiście powyższy spis właściwości nie wyczerpuje wszystkich opcji. Dostępne są również opcje pochodne, zależne od wybranych wcześniej - np. *Mod Shadow Color*, odpowiedzialny za kolor rzutowanego cienia w przypadku zaznaczenia opcji *Cast Shadows*. Można również wybrać typ cienia, jego rozdzielczość czy nawet sprawdzić jakie typy obiektów światło będzie oświetlać. Warto w ramach zapoznawania się z *UDK*, poeksperymentować z tymi właściwościami, gdyż opanowanie ich pozwoli na wprowadzanie subtelnych zmian, rzutujących na całość sceny 3D.

Podsumowując, każde światło posiada zestaw właściwości, modyfikujących algorytm propagacji światła. Do najważniejszych należą *Brightness, Color, Falloff* oraz *Shadows*.

7.4 Typologia świateł

7.4.1 Point Light

Podstawowym światłem jest światło punktowe (ang. *Point Light*) - domyślnie o białym kolorze, niezbyt dużym zasięgu i wartości *Falloff* ustawionej na 2 oraz włączonej opcji *Cast shadow*. *Point Light* można zdefiniować jako światło rozchodzące się równomiernie od źródła we wszystkich możliwych kierunkach zgodnie z algorytmem propagacji światła, uwzględniającym takie jego cechy jak natężenie, zanikanie czy rzucanie cieni. Światła punktowe mogą posłużyć, po modyfikacji ich parametrów, jako światła dopełniające (ang. *fill light*) bądź jako jaskrawe światło główne (ang. *lamp light, key light*).

Rysunek 7.6. Scena z zaznaczonym światłem punktowym - *Point light*

7.4.2 Spotlight

Drugim typem światła jest reflektor (ang. *Spotlight*), który symuluje padanie światła ze stożka o ograniczonym zasięgu. Jest to specyficzna odmiana światła, która oświetla obiekty jedynie w zasięgu zdefiniowanego wcześniej stożka. O ile światło punktowe rozchodzi się we wszystkich kierunkach, o tyle *Spotlight* tylko w kierunku, wyznaczonym przez stożek. Podobnie jak w *Point Light*, podstawowymi wartościami jest biel, niewielki zasięg i *Falloff* wynoszący 2.

Rysunek 7.7. Scena z zaznaczonym światłem typu *Spotlight*

Dodatkowo, *Spotlight* składa się z dwóch stożków, wewnętrznego oraz zewnętrznego (*inner/outer cone*), które pozwalają bardzo precyzyjnie ustawić zasięg padania światła. *Inner Cone* to wartość, opisująca stożek wewnętrzny o niezmiennym natężeniu światła, zaś *Outer Cone* to stożek zewnętrzny, który stanowi strefę, w której światło stopniowo zanika, zgodnie z wartością *falloff*.

O ile *Point Light* jest światłem, które posiada jedynie położenie w przestrzeni, opisywane za pomocą zmiennych x, y, z, o tyle *Spotlight* jest światłem, dla którego można określić rotację, również opisywaną przez trzy zmienne x, y,

z. Rotacja czyli obrót lub, mniej dosłownie, kierunek padania światła, dotyczy nie samego źródła światła a stożka.

7.4.3 Światła typu toggleable i moveable

Oba opisane wyżej typy świateł mogą być sterowane przez skrypty, stworzone w edytorze skryptów silnika graficznego *Unreal Development Kit*, zwanym *Kismet*. Światła, sterowane skryptami, nazywają się z angielskiego toggleable lub moveable. Światła typu toggleable mogą przyjmować dwie wartości - 1 i 0, odpowiedzialne za dynamiczne włączenie i wyłączenie światła. Światło typu *moveable* może się poruszać dynamicznie, zgodnie z wytycznymi ze skryptu.

7.4.4 Directional Light

Innym typem światła jest światło kierunkowe (ang. *Directional*). Światło kierunkowe nie posiada pozycji w przestrzeni 3D, tylko kierunek - nieważne zatem gdzie jest umieszczone w scenie, gdyż to właśnie ustawienie kierunku, w którym emitowane jest światło, wpływa na oświetlenie sceny. Ten typ światła pozwala na symulowanie oświetlenia słonecznego (i innych ciał niebieskich), dając wrażenie, że pada ono wprost z przestrzeni, na wszystkie elementy sceny 3D.

Rysunek 7.8. Scena oświetlona przez *Directional Light* i *Skylight* z efektami postprocesowymi

7.4.5 Komponowanie sceny z udziałem Directional Light

Jeżeli celem jest wyrenderowanie sceny, która ma mieć mocne, dzienne oświetlenie, *Directional Light* powinien być dominantą kompozycji - od jego natężenia i koloru powinna zależeć reszta świateł. Mocny, jasny kolor słonecznego po-

łudnia będzie wymagał innego oświetlenia niż krwisty zachód słońca czy różowawy poranek. Dlatego też metodą, która wydaje się być najwygodniejszą jest oświetlenie sceny najpierw za pomocą *Directional Light* o odpowiednim natężeniu, kolorze i rotacji, zaś dopiero potem dodawanie świateł typu *Point Light* lub *Spotlight*.

Podobnie też jest w przypadku komponowania scen, dziejących się w nocy - zimne i ciemne, niebieskie światło bezpośrednie stanowi swego rodzaju „podkład", który następnie powinien być rozświetlony dodatkowymi światłami.

7.4.6 Skylight

Światłem, które rozświetla wszystkie elementy sceny bez względu na ich położenie jest *Skylight*. *Skylight* to światło, które, jak sama nazwa wskazuje, pada „z nieba" (ang. *sky*), jednak pozwala też rozświetlić elementy sceny „od dołu". *Skylight* jest użyteczny między innymi wtedy, gdy chcemy stworzyć scenę, symulującą noc. Jak wyżej wspomniano, renderer nie renderuje obiektów nie oświetlonych, więc, aby cokolwiek było widać, można scenę oświetlić delikatnym światłem typu Skylight (lub *Directional Light*) o niewielkim natężeniu i kolorze zbliżonym do granatowego. W ten sposób uzyskany zostanie efekt iluminacji trójwymiarowego otoczenia mimo że, de facto, nie istnieje wyraźnie zaznaczone źródło światła, gdyż oba typy świateł nie posiadają źródła ani zasięgu. Podobnie jest w nocy, gdy ludzkie oko przystosowuje się do ciemności i widzi kontury otoczenia. *Skylight* i *Directional Light* pozwalają symulować ten proces w prosty sposób, poprzez fakt, że nie posiadają źródła ani zasięgu, więc są stosunkowo jednostajne i jednorodne. *Skylight* jest pozbawiony translacji i rotacji dlatego należy o nim myśleć raczej w kategoriach właściwości obiektu niż tradycyjnego światła ze źródłem i zasięgiem. Stanowi zatem przykład światła otoczeniowego (ambientowego).

Wymienione powyżej światła pozwalają na stworzenie wielu różnorodnych i fotorealistycznych efektów oświetlenia, poczynając od naturalnego światła słonecznego czy księżycowego, na sztucznym oświetleniu kończąc.

7.5 Światła statyczne i dynamiczne

Wszystkie dotychczas wymienione światła mogą być podzielone, bez względu na ich poprzednią klasyfikację, na światła renderowane dynamicznie i statycznie. Każde z rozwiązań ma swoje wady jak i zalety.

Światła statyczne, reprezentowane w edytorze przez literę *S*, odtwarzane są przez renderer podczas tworzenia sceny w niedokładny sposób (np. z pominięciem cieni). Dopiero uruchomienie procesu tworzenia map oświetlenia (ang. *lightmap build*) pozwala na wygenerowanie dokładnego oświetlenia, co jednak zajmuje określony i zależny od wielkości sceny, czas. Jest to podyktowane tym, że każdy obiekt w scenie zostaje „zaopatrzony" w mapę oświetlenia (mapę pro-

pagacji światła, z ang. *lightmap*) zgodną z cechami wszystkich świateł, które go oświetlają wraz z jego własnymi mapami normalnych, czyli dodatkowymi szczegółami powierzchni, które tworzone są poprzez specjalnie przygotowaną teksturę.

Światła statyczne mają swoje ograniczenia - każda zmiana nawet najmniej ważnego elementu wymaga przebudowy całości map światła, dodatkowo zaś, w procesie tworzenia sceny trudno jest o dokładny podgląd sceny. Światła statyczne nie mogą również korzystać z efektów typu *light shafts*, które symulują przechodzenie promieni światła przez zapylone powietrze, tworząc dodatkowe, zauważalne promienie. Efekt ten jest szczególnie przydatny przy scenach typu *outdoor*, czyli symulujących otwarte przestrzenie, oświetlone np. światłem słonecznym.

Zaletą świateł statycznych jest właśnie korzystanie z mapowania propagacji światła, co przy odtwarzaniu sceny nie obciąża prawie w ogóle karty graficznej komputera ani procesora i pozwala na bardzo dokładne, niemożliwe do osiągnięcia w inny sposób, odwzorowanie propagacji światła i cieni.

Światła renderowane dynamicznie nie korzystają z mapowania propagacji światła gdyż cały proces jest tworzony na bieżąco, podczas odtwarzania sceny. Pozwala to na dokładny podgląd sceny w procesie tworzenia oraz na wykorzystanie efektów takich jak *light shafts*, ma też jednak skutki uboczne, jakimi są znaczne obciążenie procesora oraz karty graficznej. Należy pamiętać, że przeciętnie skomplikowana scena może wykorzystywać kilkadziesiąt, albo i więcej świateł, co stanowi nie lada wyzwanie nawet dla współczesnych układów graficznych.

Mapy oświetlenia, jako technika stosunkowo stara, stanowi podstawę renderingu światła w wielu różnych silnikach gier. Oczywiście, możliwe jest budowanie sceny w oparciu tylko o światła liczone dynamicznie, jednakże efekt jakościowy często jest niezadowalający i zasobożerny. W skrajnych przypadkach może się okazać, że nawet stosunkowo nieskomplikowana scena przy dużej liczbie świateł dynamicznych jest niemożliwa do wygenerowania w czasie rzeczywistym lub przy założonych parametrach jakościowych.

Korzystanie z map oświetlenia jest ogromnie ważne, gdyż umożliwiają one renderowanie części oświetlenia sceny podczas procesu jej tworzenia a nie prezentacji, co jest niezwykle istotne np. w grach komputerowych, gdzie liczy się przede wszystkim wydajność oraz liczba klatek na sekundę wyświetlanego obrazu. Mapy oświetlenia (*Lightmaps*) są to dodatkowe tekstury o zadanej przez użytkownika wielkości, zachowujące dane dotyczące propagacji wszelkich świateł w scenie. Plusem takiego rozwiązania jest jego wydajność i dokładność, minusem zaś to, iż nie można generować oświetlenia zmieniającego się dynamicznie, gdyż wymagałoby to przeliczenia wszystkich map oświetlenia w czasie rzeczywistym, co jest niemożliwe. Dlatego przeważnie stosuje się rozwiązanie „pośrednie"- część świateł jest renderowana w oparciu o technikę map oświetlenia, część zaś jest generowana dynamicznie, co pozwala na kompromis pomiędzy zmiennością oświetlenia sceny i wydajnością.

Dzięki mapom oświetlenia można również wygenerować stosunkowo dokładne cienie, w tym przypadku należy jednak pamiętać o określeniu odpowiedniej rozdzielczości tych map. W przypadku BSP (czyli elementów sceny tworzonych bezpośrednio w edytorze za pomocą narzędzi do geometrii i teksturowania) rozdzielczość musi być jak najmniejsza, w przypadku siatek statycznych (*Static Meshes* - obiekty 3D, tworzone w programach graficznych i posiadające własny, nieraz bardzo skomplikowany zestaw materiałów) - jak największa. Przy czym należy tu pójść na kompromis pomiędzy wydajnością a dokładnością map oświetlenia, gdyż przypisywanie dużych rozdzielczości małym obiektom mija się z celem a zajmuje czas rendererowi oraz pochłania pamięć przeznaczoną na tekstury.

Podsumowując, UDK oferuje następujące typy świateł: *Point Light, Spotlight, Directional Light, Skylight.* Dodatkowo, światła mogą być renderowane albo statycznie (za pomocą map oświetlenia) albo dynamicznie, na bieżąco, podczas wyświetlania sceny.

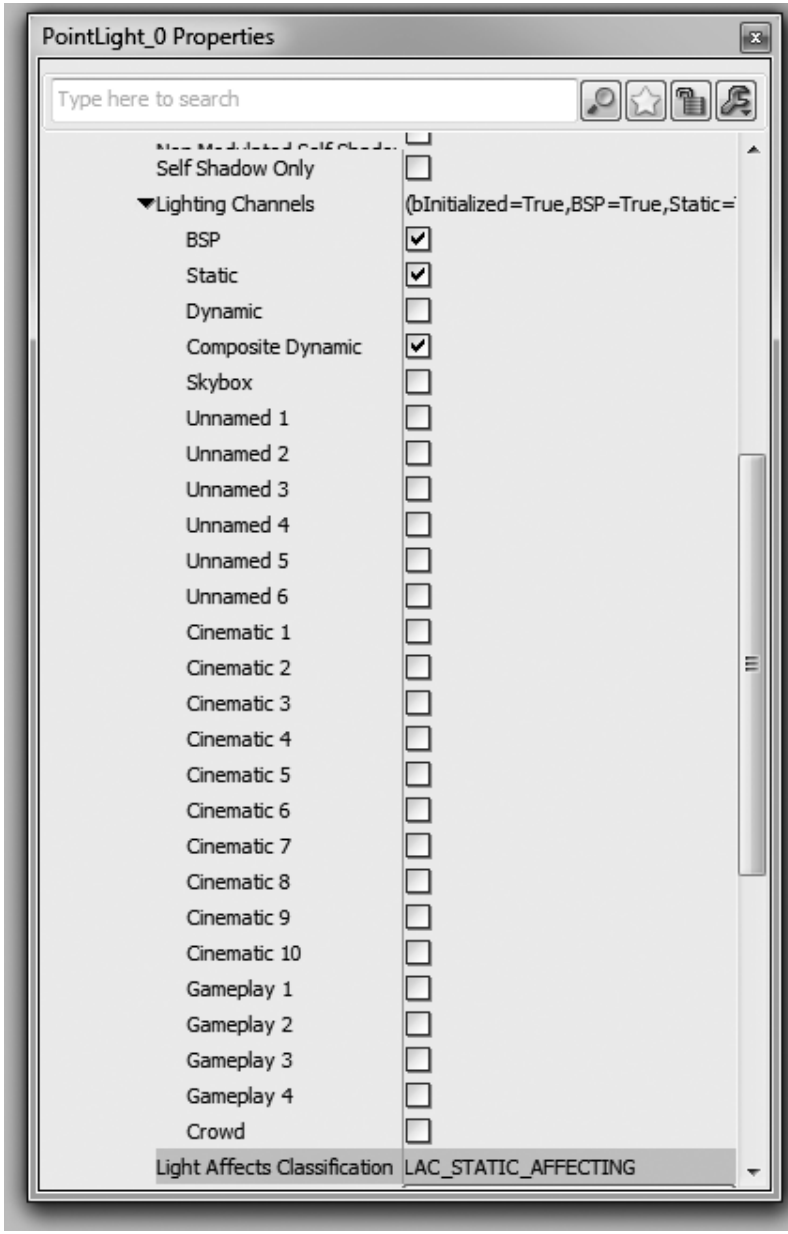
7.6 Lighting Channels

Unreal Development Kit pozwala na wykorzystanie tzw. *Lighting Channels*, czyli kanałów światła. Ta technika renderowania pozwala na wydajne i szybkie oświetlenie wybranych elementów sceny bez wpływania na inne elementy. *Lighting Channels* dostępne są we właściwościach światła, w zakładce *Light Component/Lighting Channels.*

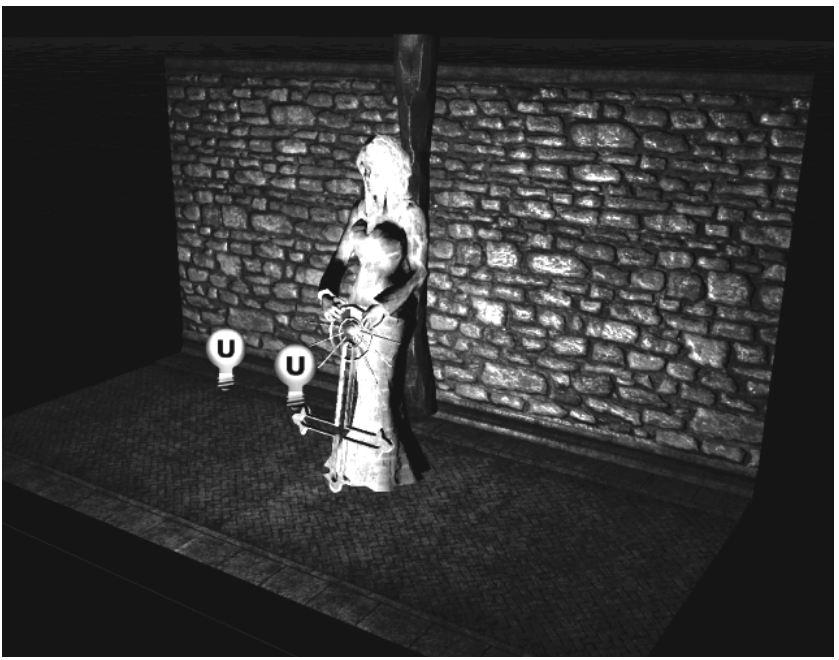
Zasada tej techniki oświetlenia jest prosta - światło musi „nadawać" na tym samym kanale, na którym „odbiera" oświetlany element, dzięki temu nie wpływa ono na inne obiekty. Przykładowo, tym kanałem może być *Static* - światło będzie w tym wypadku oświetlać tylko statyczne elementy sceny (tzw. *Static Meshes*, obiektów, przygotowanych wcześniej w dowolnym edytorze 3D i zaopatrzonych w zestaw tekstur, zwanych materiałem). W innym przypadku może świecić na obiekty animowane, czyli takie, które posiadają szkielet, służący do animacji (kanał *Dynamic* lub *Composite Dynamic*) lub na obiekty tworzone przez algorytm *BSP* na kanale *BSP.*

Na rysunku 7.9. widać zakładkę *Lighting Channels* z wybranymi kanałami - *BSP, Static* i *Composite Dynamic.* Ta ostatnia opcja pozwala na bardziej wydajne oświetlanie animowanych elementów sceny niż kanał *Dynamic.* Oprócz tego, jak widać, dostępnych jest wiele kanałów *Unnamed*, czyli niezdefiniowanych, które grafik może wykorzystać w dowolny sposób, *Cinematic*, służących do doświetlenia postaci i obiektów w przypadku, gdy służą do stworzenia animowanego filmu, a także kanały typu *Gameplay*, wykorzystywane przez elementy rozgrywki.

Podsumowując, *Lighting Channels* pozwalają na poprawienie wydajności renderowanej sceny lub podkreślanie pewnych jej elementów bez wpływania na inne. Na poniższym obrazku mocne światło przed posągiem świeci tylko na niego, natomiast słabe światło po lewej stronie - tylko na ściany i podłogę.



Rysunek 7.9. Kanały oświetlenia



Rysunek 7.10. Scena oświetlona z wykorzystaniem *Lighting Channels*

7.7 Light Function

W wielu współczesnych silnikach graficznych światło może rzucać nie tylko cienie lecz również np. symulować przejście światła przez kolorowy witraż lub inną, materialną przeszkodę (np. rozszczepienie światła w krysztale). Dzięki tej funkcji, zwanej w *UDK Light Function*, możliwe staje się, niewielkim kosztem, wzbogacenie oświetlenia o dodatkowe elementy. By tego dokonać (np. odtworzyć światło, przechodzące przez kościelny witraż i padające na podłogę), należy przygotować specjalny materiał (tzw. *Emissive Material*, czyli materiał emitujący [światło]). Materiał musi zawierać wszystkie dane, dotyczące propagacji światła i kształtów, przez to światło rzutowane. Ostatnim krokiem jest podpięcie tego materiału pod światło, przeważnie punktowe lub *Spotlight*.

7.8 Indirect Lighting i tryb renderingu Lightmass

Indirect Lightning (ang. światło pośrednie) to technika oświetlenia sceny, wykorzystująca zjawisko odbicia światła od dowolnej powierzchni. W świecie realnym, w którym rządzą prawa fizyki, każda bez wyjątku powierzchnia odbija światło (a więc wiązki fotonów), dzięki czemu jest widoczna. Przykładowo,

Księżyc nie świeci własnym światłem lecz odbitym od niewidzialnego dla nas w nocy Słońca.

Światło odbite jest dużo słabsze od światła bezpośredniego (stąd też noc jest ciemniejsza niż nawet bardzo pochmurny dzień), potrafi być jednak na tyle silne, by oświetlać także inne powierzchnie. Dlatego potocznie mówi się o świetle Księżyca, które „rozjaśnia mroki nocy" choć powinno się mówić o „świetle odbitym Słońca, rozjaśniającym mroki nocy".

UDK pozwala na symulowanie takiego światła pod warunkiem korzystania z trybu renderingu o nazwie *Lightmass*. Światło odbite ma swoje właściwości, takie jak kolor czy zasięg, nie stanowi jednak światła jako osobnej instancji, lecz jedynie pewien model propagacji.

Światło odbite pozwala na dokładne odwzorowanie propagacji, biorąc pod uwagę rodzaj powierzchni, od której światło jest odbijane. Taka technika pozwala zmniejszyć liczbę świateł w scenie, które, w innym przypadku, musiałyby „udawać", za pomocą niewielkiej wartości jasności i odpowiedniemu zasięgowi, światło odbite.

Lightmass jest specjalnym trybem renderingu oświetlenia - generuje oświetlenie za pomocą algorytmu, który emituje teoretyczne fotony (traktowane tu jako cząstki światła, choć samo światło ma naturę falowo-korpuskularną), bada ich odbicie od dowolnej powierzchni i na podstawie zebranych danych generuje oświetlenie. Jest to proces dużo dokładniejszy od renderingu map oświetlenia wyłącznie w oparciu o światło bezpośrednie. Niestety wymaga on dużo więcej czasu i mocy procesora. Aby ułatwić przeliczanie skomplikowanych scen w silniku Unreal zastosowano przetwarzanie rozproszone, które tutaj nazywa się *Swarm* (ang. rój). Dzięki niemu jeden z połączonych w sieć LAN (*Large Area Network*) komputerów staje się koordynatorem, przydzielającym zadania innym komputerom, które obliczają wspólnie oświetlenie sceny.

Na pierwszy rzut oka *Lightmass* wydaje się narzędziem ociężałym i zajmującym dużo czasu procesorowi, jednak w przypadku korzystania z przetwarzania rozproszonego i kilku komputerów jakością oświetlenia w skomplikowanych scenach bije na głowę inne techniki renderowania świateł.

7.9 Efekt postprocesowy Bloom i rożne rodzaje mgieł

Efekt postprocesowy to efekt, który renderer „nakłada" na już gotową, wyrenderowaną scenę w dodatkowym przebiegu, co oznacza, że np. jedna klatka gry, zanim zostanie przez renderer przekazana do urządzenia wyjścia (np. wyświetlona na ekranie monitora), musi zostać jeszcze raz „przerobiona" przez renderer. Łatwo zauważyć, że dodatkowo obciąża to procesor przy liczeniu sceny, dlatego, podobnie jak w przypadku cieni, efekty postprocesowe powinny być używane z umiarem. Łatwo tu popaść w przesadę i całkowicie zniszczyć wypracowany oświetleniem efekt, gdyż w przypadku postprocesow więcej nie znaczy lepiej.

Zatem, w pierwszym przebiegu renderer tworzy scenę, uwzględniając w niej propagacje światła, wygenerowane wcześniej *lightmapy* oraz setki innych elementów, które składają się na scenę 3D. Dopiero po jej przygotowaniu, renderer generuje efekty postprocesowe, nie przekazując od razu obrazu na wyjście tylko jeszcze raz przetwarzając go zgodnie z zadanymi parametrami. Dodatkowych „przejść" (ang. *pass*) renderera może być więcej niż jedno - wiele zależy tu od liczby postprocesów oraz ich parametrów. Dopiero po zakończeniu wszystkich „przejść" obraz wysyłany jest na urządzenie wyjścia.

Projektując oświetlenie należy również pamiętać o wkomponowaniu odpowiednich efektów postprocesowych, uwzględniając także dodatkowe przejścia renderera. Być może w ostatecznym rozrachunku okaże się, że z pewnych elementów sceny należy zrezygnować, bowiem zbyt obciążają procesor, światło zaś przekomponować i na nowo wygenerować mapy oświetlenia lub zwiększyć zmniejszyć liczbę postprocesów.

Rysunek 7.11. Scena z efektami postprocesowymi *DOF* i *Bloom*

7.10 Bloom

Bloom jest jednym z częściej używanych efektów. Polega na rozmyciu (stworzeniu charakterystycznej „mgiełki") wokół świateł charakteryzujących się dużą energią lub powierzchni rozświetlonych mocnym światłem. Dzięki temu efektowi plamy światła stają się miększe, zacierają się granice pomiędzy światłem a powierzchnią, na którą owo światło rzutuje

Scena z efektami typu *bloom*

Rysunek 7.12. Scena bez efektu typu *bloom*

Właściwości efektu *Bloom* są rożne w rożnych silnikach graficznych, warto jednak pamiętać, że w każdym z nich można ustawić jego natężenie. Zgranie natężenia tego efektu z ogólnym nastrojem sceny z takimi elementami jak np. mgła (ang. *Fog*), natężenie światła dominującego czy chociażby symulowana pora dnia jest niezbędne, by osiągnąć pożądany efekt. Przykładowo, w outdoorowej scenie z mocnym światłem silny *Bloom* da wrażenie „gorąca" (zwłaszcza w połączeniu z innymi efektami postprocesowymi); jeżeli do tego dojdzie odpowiednia kolorystyka dominującego światła, łatwo można osiągnąć efekt rozgrzanej, parującej pustyni, zalanej silnym, rażącym wręcz słonecznym światłem.

Innym przykładem może być światło nocne, które ma kolorystykę zbliżoną do naturalnego blasku księżyca. W połączeniu z lekkim bloomem można osiągnąć ciekawy efekt jarzenia się niektórych, oświetlonych powierzchni.

W obydwu przypadkach, świetle dziennym i nocnym, to *Bloom* nadaje końcowy efekt jasnym powierzchniom. Oczywiście, nie jest on czymś absolutnie niezbędnym, stanowi jednak bardo dobre narzędzie do „dopełniania" skomponowanego oświetlenia.

W scenach indoorowych (lecz nie tylko) *Bloom* idealnie współgra z efektem mgły (ang. *Fog*). *Fog* nie jest efektem postprocesowym lecz, zależnie od renderera, np. siatką o odpowiednim materiale. W UDK jest kilka rodzajów mgły - *Volumetric Fog, Spherical Fog, Height Fog* i *Exponential Height Fog*. Pierwsze dwa służą do „wypełnienia" pomieszczeń mgłą o takich modyfikowalnych parametrach jak kolor czy natężenie, dwa następne służą do wypełnienia całości sceny jednolitą mgłą o modyfikowalnych parametrach. Dodatkowo, *Exponential Height Fog* pozwala na umieszczenie kamery na dowolnej wysokości w scenie, gdyż nie znika w ostry sposób lecz łagodnie przechodzi w inny, zdefiniowany przez użytkownika kolor, pozwalając zgrać mgłę z tzw. skyboxem, symulującym niebo i chmury.

Zatem, by uzyskać wrażenie mglistego poranka, należy dopasować kolorystykę trzech elementów - światła dominującego w scenie, mgły oraz blooma. W przypadku wykorzystania mgieł o skończonym zasięgu jak *Volumetric Fog* lub *Spherical Fog*, należy także pamiętać o ich zgraniu z najbliższym światłem.

Ogólnie rzecz ujmując, zarówno światła jak i mgła czy *Bloom* powinny mieć zbliżoną kolorystykę, gdyż symulują kilka, powiązanych ze sobą zjawisk fizycznych, takich jak rozproszenie światła w zawiesinie wodnej.

7.10.1 Postprocess Volume

Postprocess Volume jest to zestaw postprocesów, które można w łatwy sposób przypisać do danego obszaru sceny. Wystarczy stworzyć obiekt typu *Brush* i kliknąć ikonę *Add/Posptrocess Volume* z zaznaczonym obiektem typu *Brush*. Dzięki temu w łatwy sposób można włączać i wyłączać efekty takie jak *Bloom, Dof* (ang. *depth of field*, głębia ostrości) czy zmieniać barwy wszystkich elementów w zasięgu postprocess volume za pomocą opcji *Scene_ Midtones, Sce-*

Scena z efektem mgły

Rysunek 7.13. Scena bez efektu mgły

ne_Hightones czy *Scene_Shadows*. Dzięki temu światła mogą bezproblemowo zmienić swoją barwę, pozwalając na łatwe wprowadzanie zmian bez potrzeby dodatkowego tworzenia na nowo mapowania propagacji świateł. Dodatkowo można użyć efektu, nazywającego się *Tonemapper*, który odpowiada za różnice tonalne między najjaśniejszym i najciemniejszym punktem sceny. Zwiększenie wartości tego efektu sprawi, że cienie wydadzą się głębsze zaś miejsca jasne - bardziej jaskrawe.

7.10.2 Light Volume

Light Volume to narzędzie w UDK, pozwalające na manewrowanie zasięgiem światła w inny sposób niż za pomocą parametrów czy kanałów. Pozwala uniknąć najczęstszego z błędów - przenikania światła przez powierzchnie, poprzez które światło nie może przenikać lub ograniczyć padanie światła do konkretnego kształtu. Dzięki temu blask pochodni nie przebije się przez mur, który jest zbudowany z cegły i nie przepuszcza światła.

Aby to osiągnąć, należy, wykorzystując narzędzia do pracy z geometrią sceny, zbudować niewidzialny obiekt zwany *Brush* i określić go jako jeden z typów *Light Volume: Exclude Light Volume* - który jest niewidzialną przeszkodą dla źródła światła, zapobiegając jego rozchodzeniu się lub *Light Volume* - który powoduje, że światło rozchodzi się tylko w zasięgu stworzonego brusha.

Light Volume służy do poprawiania błędów w oświetleniu bez potrzeby jego przekomponowywania. Dla użytkownika końcowego jest całkowicie niewidoczny i niewyczuwalny, pozwala jednak na stosowanie mocnych świateł w pobliżu obiektów statycznych (*Static Meshes*), wyłączając przestrzeń za obiektami z oświetlenia.

7.10.3 Light Environment

Light Environment to kolejne narzędzie, ułatwiające pracę w bardziej skomplikowanych scenach. Nie ma wpływu na bezpośredni wygląd sceny, pozwala jednak na przechowywanie podstawowych właściwości dla rożnych typów świateł (statycznych i dynamicznych), automatycznie w razie potrzeby przypisując te właściwości nowo powstałym światłom.

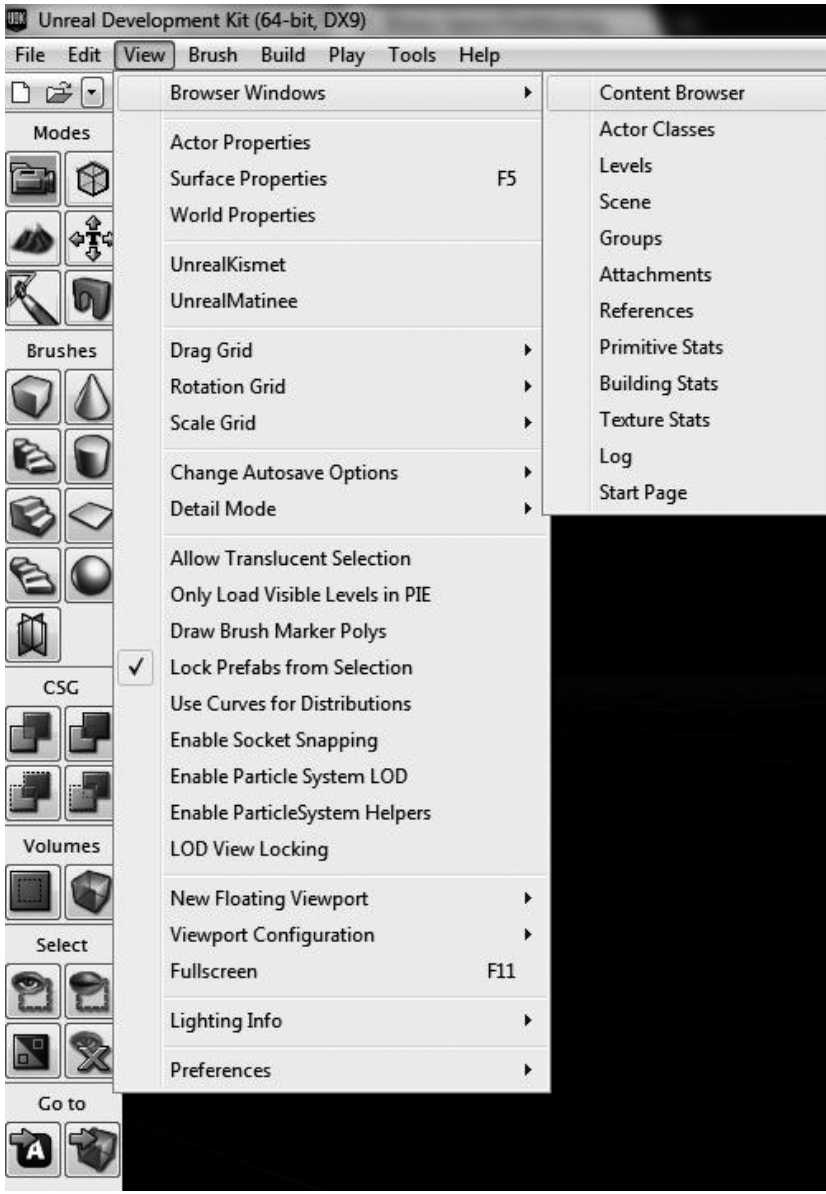
7.11 Praktyczne projektowanie oświetlenia - dodawanie świateł do sceny

Aby móc dodawać światła do sceny, należy, z menu *View* wybrać *Content Browser* (ang. przeglądarka kontekstowa), następnie zakładkę *Actor Classes* (ang. klasy aktorów, bowiem wszystkie elementy sceny są określane w *UDK* jako actors), jak na rysunku 7.14.

Lista *Actor Classes* zawiera wszystkie klasy obiektów, które można umieścić w scenie 3D (rysunek 7.15).

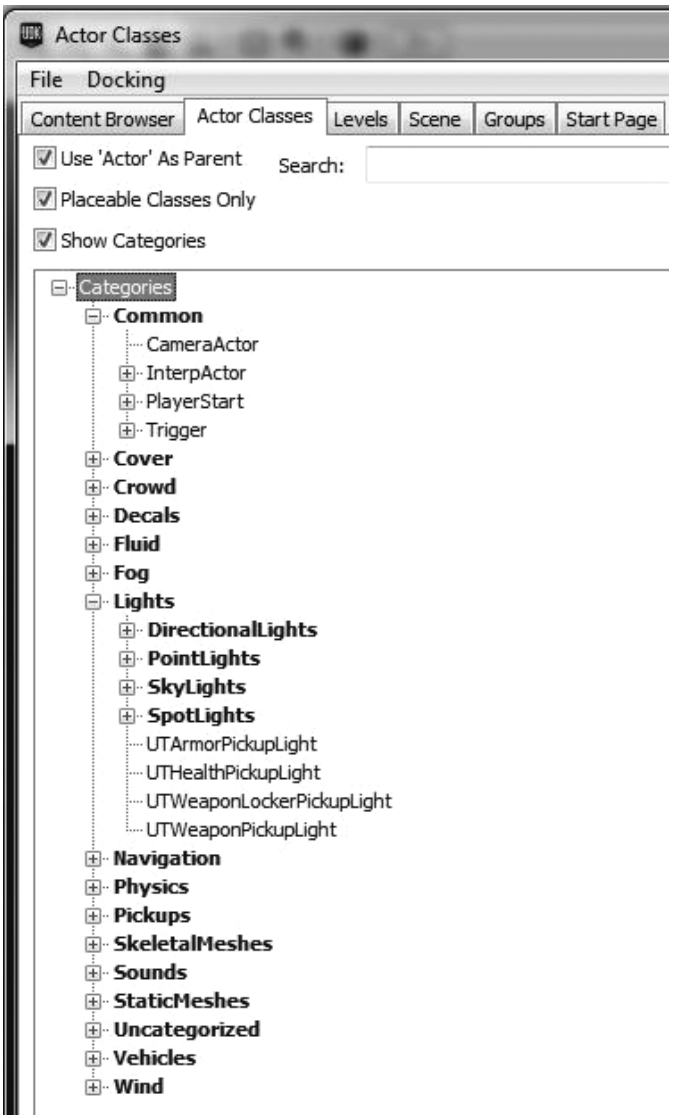
Z listy *Lights* należy wybrać dowolny typ światła, które zostanie automatycznie wstawione w scenie 3D. Jeżeli nie został wybrany żaden obiekt w scenie, światło zostanie umieszczone w tym samym miejscu, w którym znajduje się kamera, za pomocą której obserwujemy scenę 3D w edytorze.

Światło, umieszczone w scenie, oznaczone jest przez zieloną ikonę żarówki oznaczoną *S, D/S* lub *U* (od angielskich słów *static, dynamic/static* i *user changed*). Litera przy ikonie oznacza typ światła: *S* - statyczne, *D/S* - światło dy-



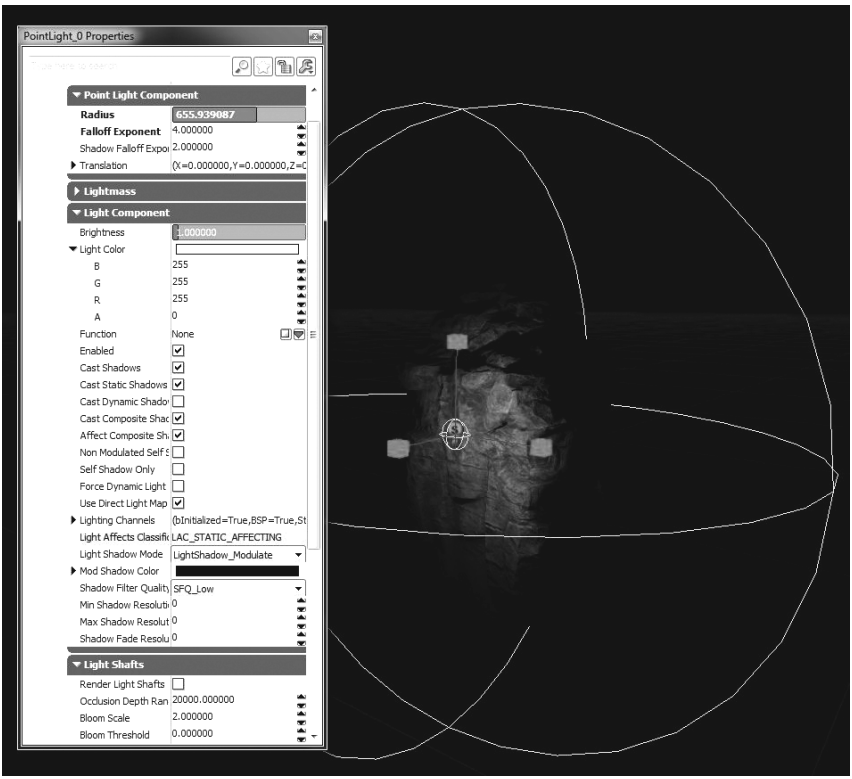
Rysunek 7.14. Otwieranie przeglądarki obiektów *Content Browser*

namiczne zaś U - światło zmienione przez grafika w szczególny sposób, np. korzystające z nietypowego kanału.



Rysunek 7.15. Widok panelu *Actor Classes*

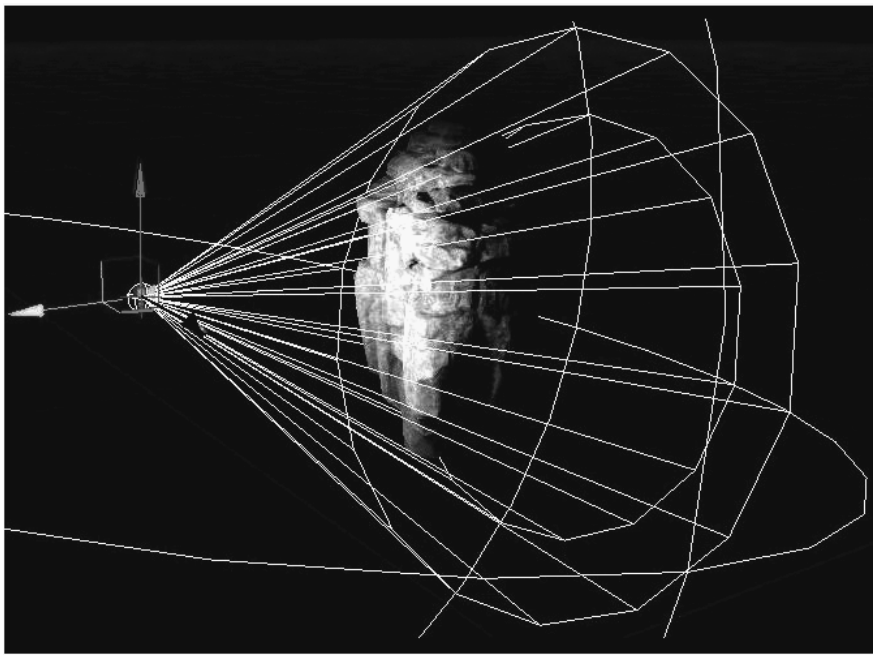
Na rysunku 7.16 obiekt typu *Static Mesh* oświetlany jest przez statyczne światło punktowe (*Point Light*). Z boku widoczne są właściwości światła. Zakładka *Point Light Component* zawiera takie parametry, jak: *Radius, Falloff* czy *Translation*, natomiast *Light Component* pozwala na korekcję natężenia światła, zmianę koloru, przypisanie mu *Light Function* czy też zmianę właściwości cieni. W tej zakładce można modyfikować również *Lighting Channels* a także



Rysunek 7.16. Światło typu *Point Light* i jego właściwości

wymusić na renderer traktowania światła jako dynamiczne (*Force Dynamic Light*). Dodatkowo ale już poza zakładką, dostępna jest funkcja *Light Shafts*, która jednak pozostanie bez wpływu na otoczenie 3D póki światło nie zostanie zaznaczone jako dynamiczne.

Ikona światła posiada trzy elementy, odpowiadające trzem osiom współrzędnych, które, po naciśnięciu spacji, zmieniają swoje funkcje, pozwalając odpowiednio na edycję zasięgu światła, jego pozycji w przestrzeni 3D oraz rotacji (co w przypadku niektórych typów świateł nie ma znaczenia - na przykład obrót światła punktowego nie da żadnego widocznego efektu). Linie, zakończone czerwonymi sześcianami służą do operacji zwiększania i zmniejszania zasięgu; zakończone strzałkami - do przesuwania emitera światła w przestrzeni 3D, zaś okręgi wokół światła do zmiany rotacji światła. Wszystkie te operacje są dostępne również z głównego menu programu, można je także wykonać poprzez wpisywanie odpowiednich wartości liczbowych we właściwościach światła. Ten ostatni sposób jest wprawdzie najmniej wygodny, lecz pozwala na dokładne zarządzanie wszystkimi właściwościami edytowanego światła.



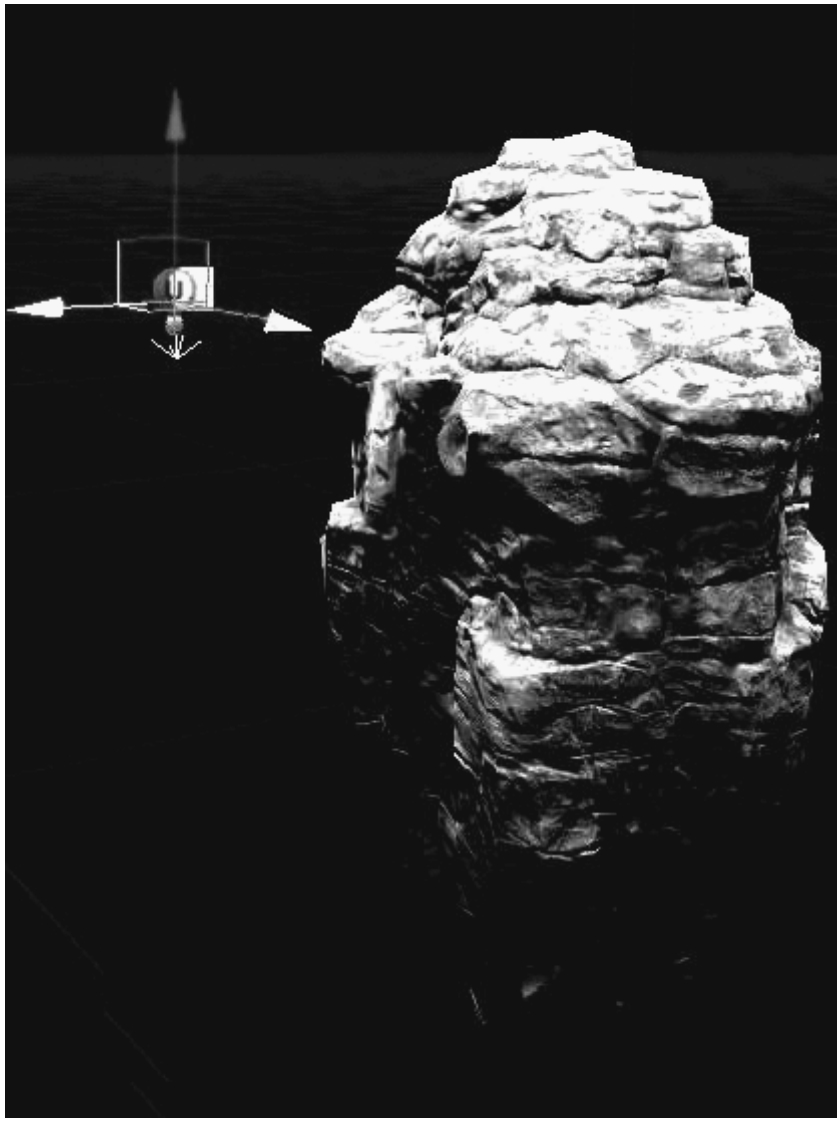
Rysunek 7.17. Światło typu *Spotlight*

Na rysunku 7.17 pokazano światło typu *Spotlight*, oświetlające obiekt statyczny od lewej strony. Jak widać, obiekt pozostaje w zasięgu wewnętrznego stożka światła (*Inner Cone*) zaś jego najbardziej wysunięte elementy wchodzą w zasięg zewnętrznego stożka (*outer cone*), przez co uzyskano wrażenie miękkiego zanikania światła (wartość *Falloff* ustawiona została na 6).

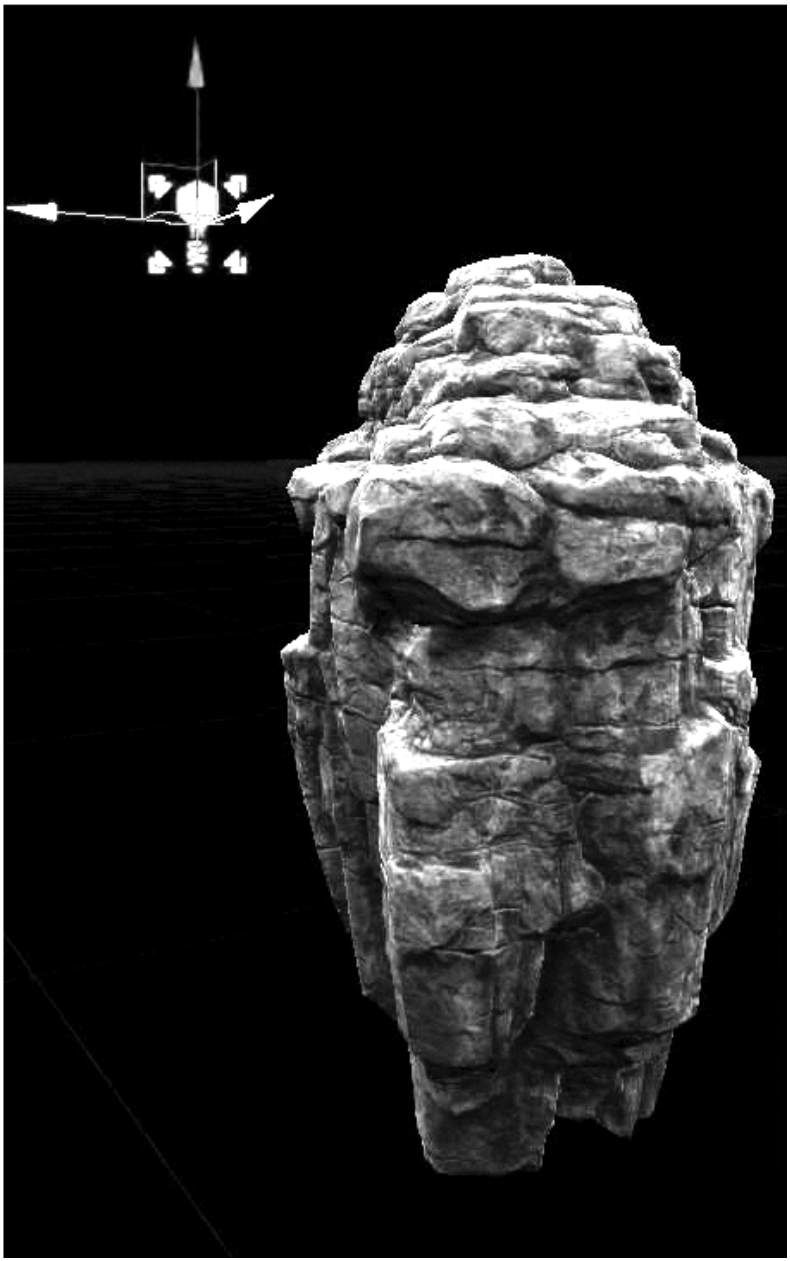
Dodatkowe białe okręgi wyznaczają zasięg światła, jednak nie mają zasadniczego znaczenia, gdyż tę samą funkcję pełni, zobrazowany za pomocą prostego obiektu, stożek.

Światło z rysunku 7.18 to *Directional Light*; jak widać, mimo że ikona światła znajduje się po lewej stronie obiektu, mamy wrażenie, jakby światło świeciło z góry. Dodatkowo, ikona, oprócz strzałek, pozwalających na edycję położenia światłą w przestrzeni sceny, posiada białą, wieloramienną strzałkę, oznaczającą kierunek padania światłą, gdyż jego pozycja jest stała natomiast rotacja może się zmieniać. Światło to nie ma zasięgu ani translacji, więc edycja tych właściwości nie wpływa na światło w żaden sposób.

Na rysunku 7.19 przedstawione zostało światło typu *Skylight* - znów, mimo że podobnie jak *Directional Light* umieszczono je z boku obiektu, świeci prostopadle z góry. *Skylight* nie posiada ani translacji ani rotacji, więc edycja tych wartości nie przynosi żadnych efektów.



Rysunek 7.18. Światło typu *Directional Light*

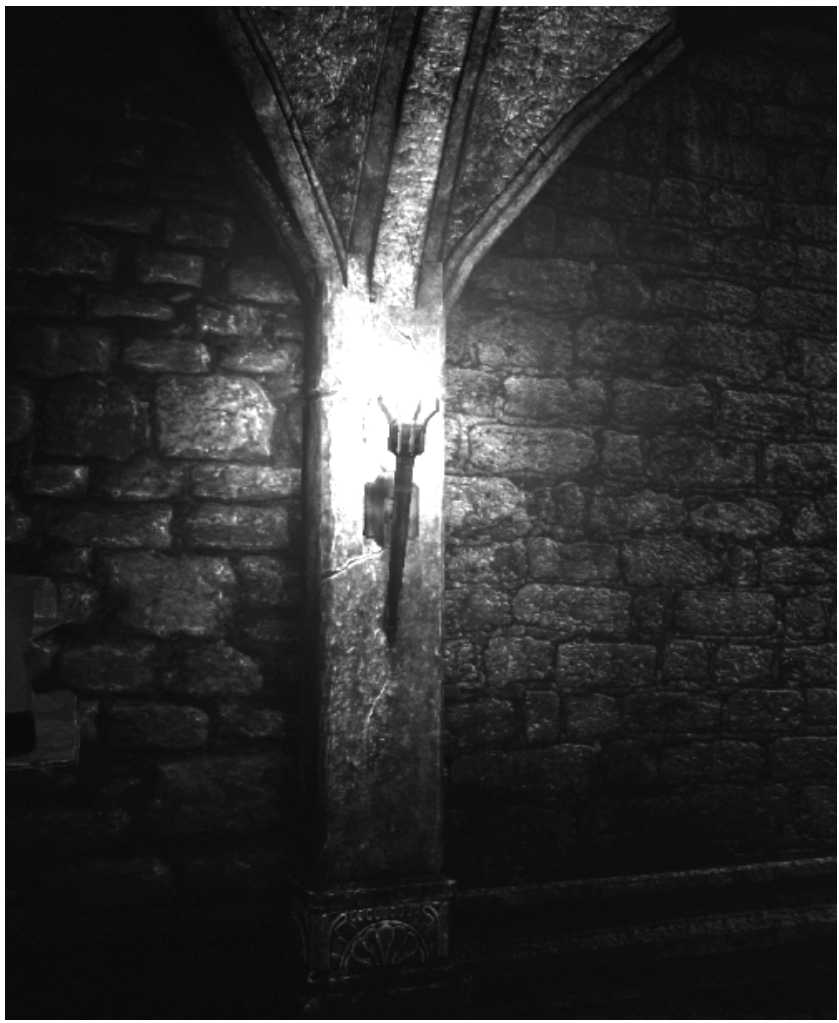


Rysunek 7.19. Światło typu *Skylight*

7.12 Uzupełnianie świateł efektami

Światła w silniku UDK można wzbogacić następującymi efektami:

- Flary - to obiekty, symulujące za pomocą specjalnego materiału rozbłysk światła o dużej energii (a więc jasności) i jego załamanie w ośrodku atmosferycznym (charakterystyczne „promienie") - rysunek 7.20.



Rysunek 7.20. Scena z wykorzystaniem *flary* i efektu cząstkowego *particles*

- Efekt wolumetryczny - efekt przechodzenia światła przez mocno zanieczyszczone powietrze. O ile w innych silnikach graficznych efekt ten jest przeważnie tworzony proceduralnie (np. w *Trinigy Vision*) o tyle w *UDK* należy wykorzystać przygotowany wcześniej obiekt i materiał, który symuluje przechodzenie światła przez zanieczyszczenia (rysunek 7.21).

Rysunek 7.21. Scena wykorzystująca efekt wolumeryczny

- *Bloom* - postprocess, który pozwala dodać rozświetlenie do wszystkich mocno oświetlonych powierzchni.

Rysunek 7.22. Scena z efektem *Light Shafts*

- *Light Shafts*, zwane też czasem *god rays* (ang. boskie promienie) - doskonale współgrające z efektami takimi jak *Bloom*, efektem wolumetrycznym czy flarami (rysunek 7.22).

8

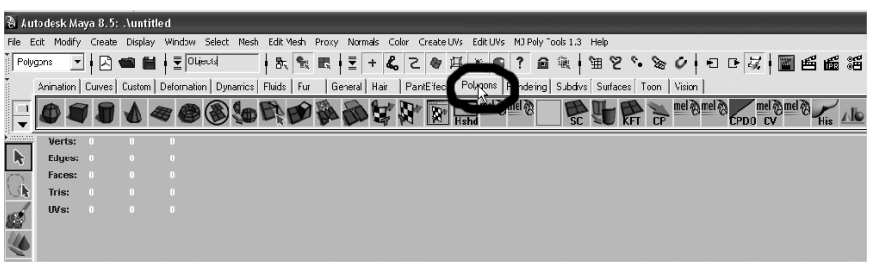
Podstawy animacji w programie Maya

8.1 Wprowadzenie

Animacja 3D opiera się na, mówiąc w uproszczeniu, ruchu obiektów scenie 3D. Istnieje wiele rodzajów i sposobów animacji komputerowej. W tym rozdziale skupimy się na przykładach animacji 3D tworzonej w programie Maya firmy Autodesk. Metody, które zostaną tu opisane, to animacja kinetyczna - czyli proste przesuwanie i obracanie obiektów w określonym czasie - oraz podstawy animacji szkieletowej, która służy do odkształcania bardziej skomplikowanych trójwymiarowych modeli.

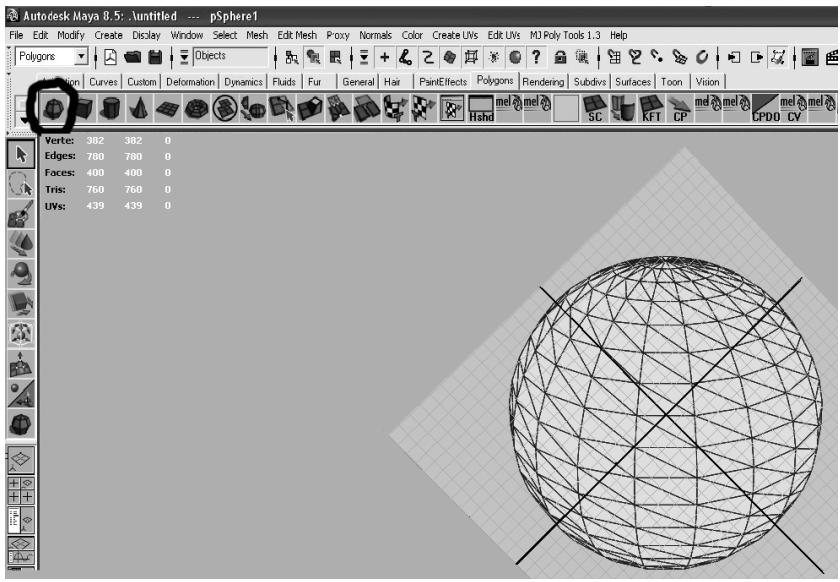
Pierwszą animacją, którą tu wykonamy będzie tocząca się kula rozbijająca kręgle. Na początek stworzymy obiekty do animowania - w tym przypadku kulę i trzy kręgle (w naszym obiekcie reprezentowane przez cylindry, lecz czytelnik może wymodelować samodzielnie bardziej złożone bryły). W poniższych opisach wykorzystania narzędzi programu Maya skrót LKM oznaczać będzie Lewy Klawisz Myszy, ŚKM - Środkowy Klawisz Myszy, a PKM - Prawy Klawisz Myszy.

Aby wymodelować kulę, na listwie narzędzi kliknijmy zakładkę *Polygons*. Jej położenie jest zaznaczone na rysunku 8.1.



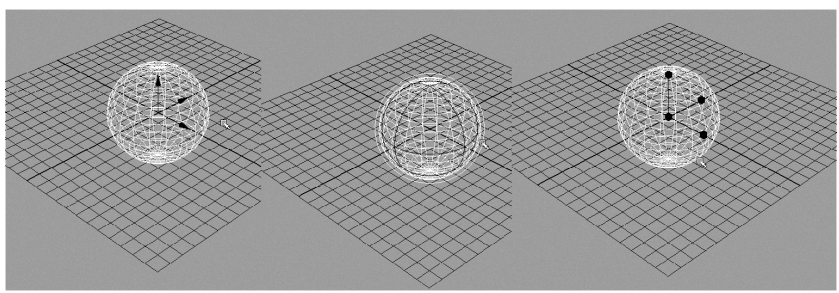
Rysunek 8.1. Zakładka Polygons

Na listwie narzędzi pierwszą pozycją po lewej jest tworzenie sfery (*Polygon Sphere*) czyli trójwymiarowej kuli. Klikamy na ikonie sfery (por. rysunek 8.2), przenosimy kursor nad okno widokowe i trzymając LKM przeciągamy kursorem tak, żeby kula osiągnęła odpowiedni rozmiar. Gdy to nastąpi, zwalniamy LKM.



Rysunek 8.2. Tworzenie kuli

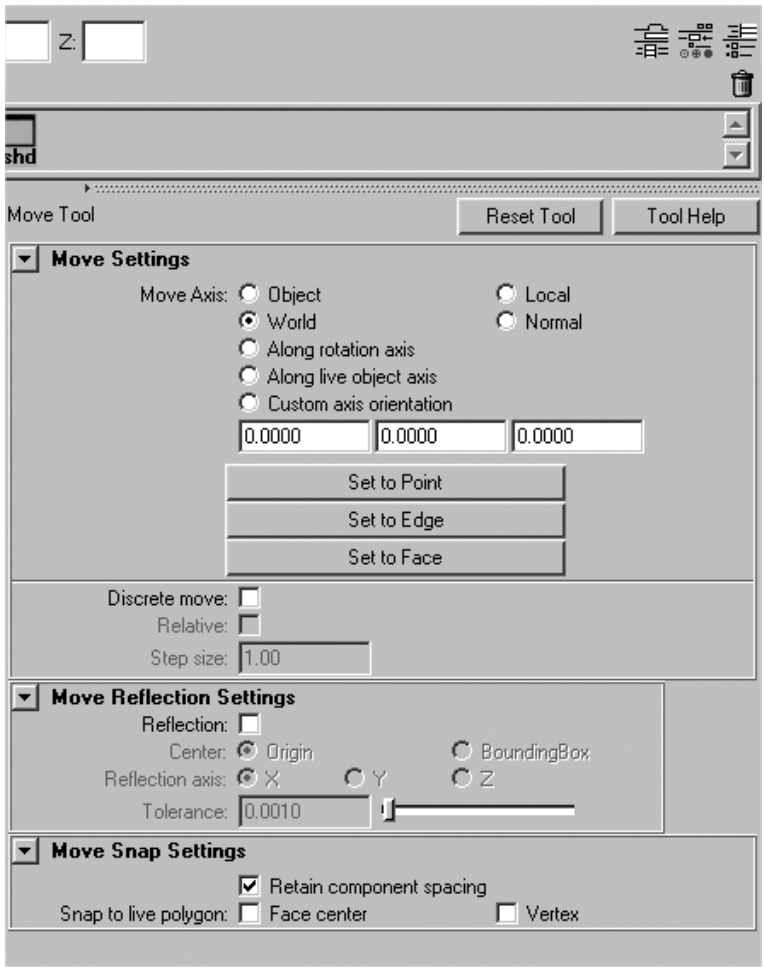
W programie Maya istnieją trzy podstawowe tryby transformowania obiektów, których sprawne wykorzystanie jest niezbędne do animacji (rysunek 8.3). Jest to tryb przesuwania obiektów, włączany klawiszem W, obracania obiektów - klawisz E oraz skalowania obiektów - klawisz R. Aby powrócić do trybu wybierania obiektów, naciskamy klawisz Q.



Rysunek 8.3. Narzędzia transformacji przesuwania, obracania i skalowania

Mając włączony tryb przesuwania, możemy przemieszczać obiekt swobodnie względem aktywnego widoku, poprzez kliknięcie oraz przytrzymanie LKM w środku ikony transformacji (żółty kwadrat) przesuwając myszą w dowolnym kierunku. Jeśli chcemy przesuwać obiekt tylko po wybranej osi (X, Y, Z), klikamy i przytrzymujemy LKM na strzałkach które rozchodzą się ze środka ikony transformacji i poruszamy myszą w wybranym kierunku.

Adekwatnie do trybu przesuwania, w trybie obracania (wizualizowanym jako kula z oznaczonymi osiami), możemy obracać obiekt po wybranej osi (przytrzymując LKM na obwodach narzędzia o odpowiednich kolorach - oś X to czerwony kolor, Y zielony, Z niebieski), lub też w dowolnym kierunku, klikając LKM pomiędzy oznaczonymi obwodami.



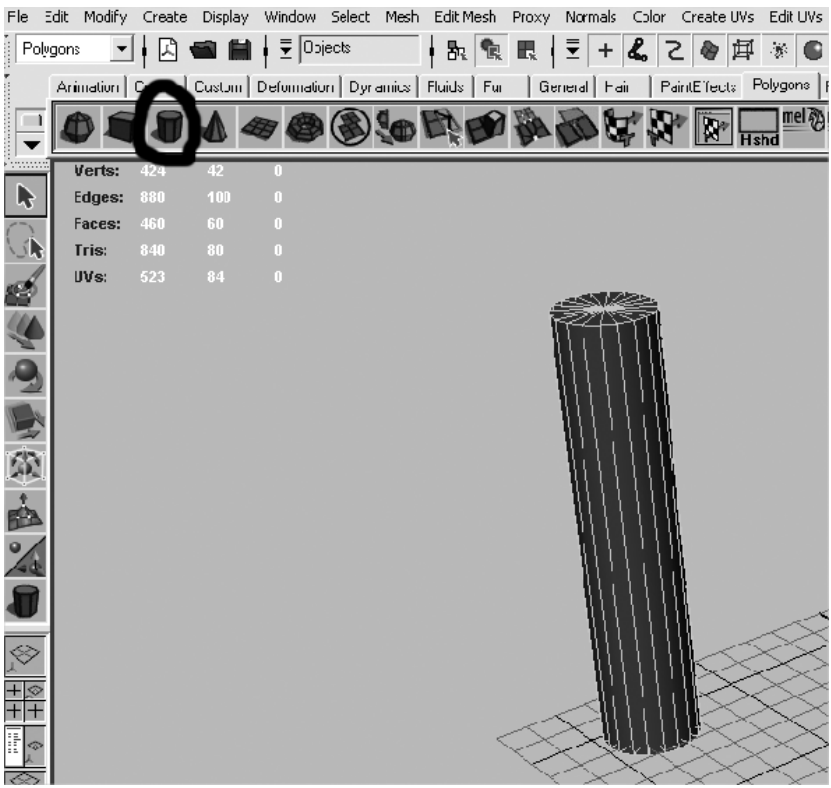
Rysunek 8.4. Opcje narzędzi transformacji

Tryb skalowania działa podobnie do trybu przesuwania - trzymając LKM i przeciągając mysz na żółtym centrum narzędzia - skalujemy cały obiekt. Postępując tak samo na wybranych osiach, skalujemy obiekt tylko wzdłuż odpowiedniej.

Aby powrócić do trybu zaznaczania, naciskamy klawisz Q.

Wspomniane narzędzia mają swoje dodatkowe opcje. Okno opcji narzędzi uzyskujemy przez naciśnięcie LKM na ikonie *Tool Settings* wyróżnionej na rysunku 8.4. Tu najważniejsze są opcje do wyboru osi transformacji (*Move Axis*), w szczególności tryby *Object* i *World*. Gdy wybierzemy *Object*, narzędzia będą działać z zaznaczonym obiektem jako punktem odniesienia, natomiast przy wybranej opcji *World*, punktem odniesienia jest cała scena. Dobrze jest przetestować praktyczne różnice pomiędzy tymi opcjami, wybierając którąś z nich i sprawdzając, jak zmienia się działanie narzędzi przesuwania i obracania.

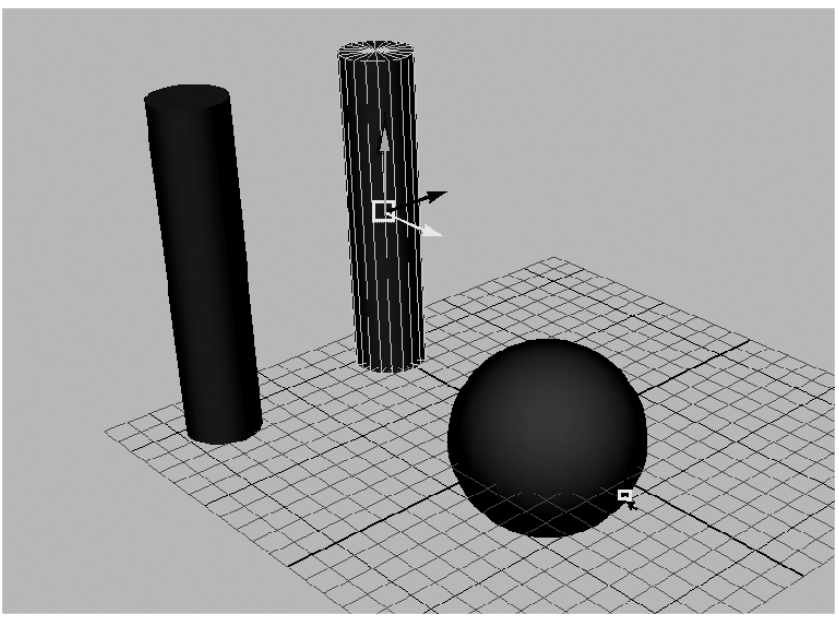
Następnymi obiektami, potrzebnymi do stworzenia naszej animacji są kręgle, a właściwie - trójwymiarowe cylindry. Tworzymy je wybierając z listwy narzędzi trzecią ikonę od lewej (rysunek 8.5), po czym przenosimy kursor nad ok-



Rysunek 8.5. Tworzenie cylindra

no widokowe, klikamy i trzymamy LKM, tworząc podstawę cylindra o właściwym rozmiarze, puszczamy LKM, naciskamy i przytrzymujemy go ponownie, aby ustawić odpowiednią wysokość bryły.

Aby wszystkie nasze „kręgle" były tych samych wymiarów, zamiast tworzyć pozostałe bryły od nowa - skopiujemy już istniejący. W tym celu, mając zaznaczony pierwszy cylinder, naciskamy kombinację klawiszy SHIFT+D, co utworzy nam się identyczny cylinder w dokładnie tym samym miejscu co pierwotna siatka. Wchodzimy w tryb przesuwania (W) i przeciągamy nowy obiekt po osi Y (czarna strzałka). W efekcie powinniśmy uzyskać 2 kręgle obok siebie, jak pokazano na rysunku 8.6.

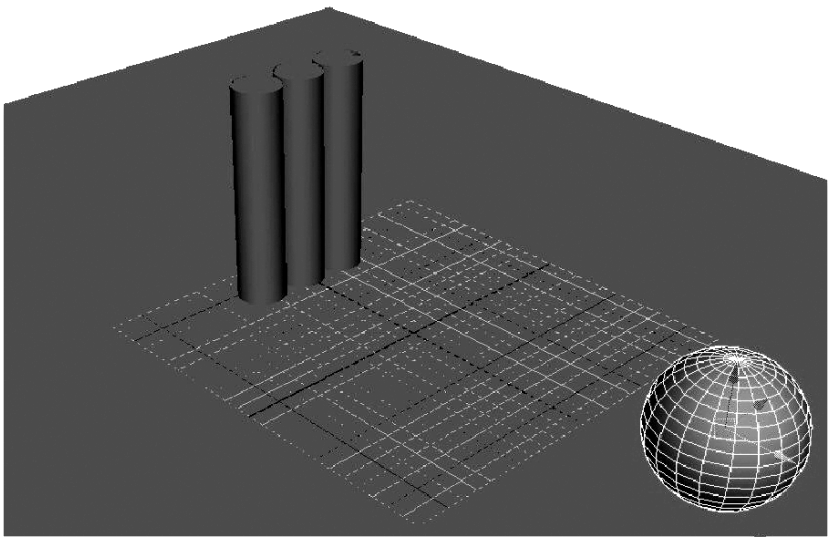


Rysunek 8.6. Kopiowanie obiektów

Mając zaznaczony drugi cylinder, kolejny raz wciskamy klawisze SHIFT+D. Kolejny cylinder powinien utworzyć się nam w tej samej odległości co drugi od pierwszego (jeżeli nie, to trzeba go ręcznie przesunąć).

Na scenie znajdują się teraz trzy cylindry, ustawiamy je bliżej siebie (tak, żeby kula mogła je wszystkie rozbić). Cały czas brakuje nam podłoża do toczenia kuli animacji. Aby go utworzyć, wybieramy z listwy narzędzi piątą pozycję (obiekt *Plane* - płaską powierzchnię). Tworzymy podłoże klikając i przytrzymując LKM w oknie widokowym i przeciągając mysz w odpowiednim kierunku. Jeżeli utworzony obiekt jest niewłaściwych rozmiarów lub w nieodpowiednim położeniu, używamy trybu przesuwania i skalowania, aby dopasować go do naszych potrzeb. Ostatecznie pozostaje nam podniesienie kuli tak, aby

„leżała" na podłożu oraz odsunięcie jej, tak, żeby mogła toczyć się przez jakiś czas zanim trafi w kręgle. Przykładowy wygląd sceny na tym etapie pracy pokazano na rysunku 8.7.



Rysunek 8.7. Scena przygotowana pod animację

8.2 Zasady działania animacji kluczowej

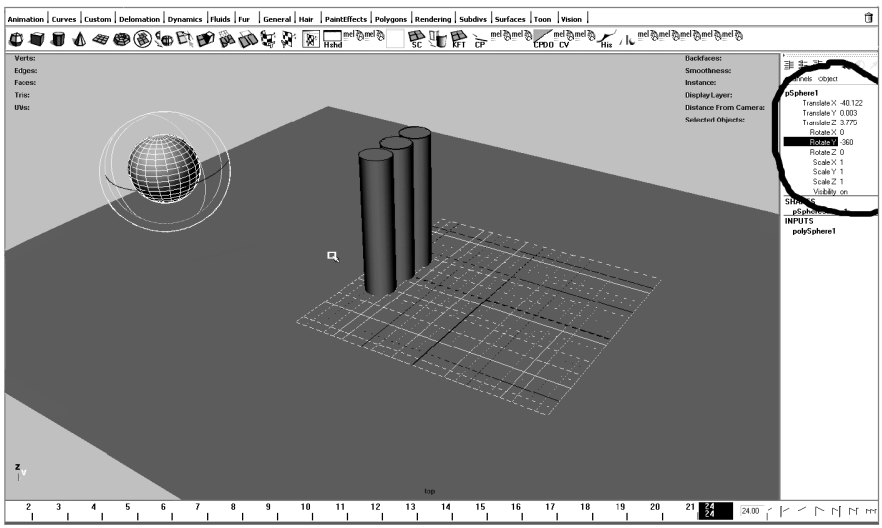
Animacja kluczowa polega na interpolacji atrybutów danego obiektu (takich jak przesunięcie, obrót czy skala) z jednej wartości do drugiej w pewnym czasie. Dokonuje się tego tworząc tzw. „klucze" (klatki kluczowe, ang. *keys*), oznaczające kluczowe pozycje obiektu w odpowiednich odstępach czasowych. Do przemieszczania się w czasie służy nam listwa czasu (*Timeline*), znajdująca się centralnie pod oknem widokowym. Cyfry na listwie czasu oznaczają numery klatek animacji (zwykle 24 klatki to jedna sekunda). Czarny prostokąt na listwie czasu oznacza, w której klatce animacji wyświetlana jest zawartość okna widokowego. Aby zmienić bieżącą klatkę, klikamy w dowolnym miejscu listwy czasu za pomocą LKM.

Do tworzenia kluczy służy klawisz S, który naciskając zapisujemy transformacje obiektu dla wybranej klatki.

8.2.1 Animacja kluczowa z użyciem osi czasu oraz edytora krzywych

Czas zacząć pracę nad samą animacją. Będąc w trybie zaznaczania i mając listwę czasu ustawioną na pierwszej klatce, naciskamy LKM na naszej kuli. Sta-

wiamy klucz (klawisz S) - na listwie czasu powinna pojawić się czerwona kreska oznaczająca, w której klatce znajduje się postawiony klucz. Następnie klikamy LKM w 24-tej klatce animacji na listwie czasu, żeby przenieść się do tego momentu w czasie. W trybie przesuwania (klawisz W), przestawiamy kulę mniej więcej na koniec naszego podłoża po osi X (czerwona strzałka). Aby kula sprawiała wrażenie „toczącej się", trzeba ją obrócić o odpowiednią liczbę stopni w osi Y. W trybie obracania, po osi Y (zielonym obwodzie) obracamy kulę tak, by w polu atrybutów parametr Rotate Y wyniósł -360. Klikając LKM w polu atrybutu, można wpisać ją ręcznie. Edytor atrybutów wyróżniony jest na rysunku 8.8. W klatce 24. również tworzymy klucz transformacji (klawisz S).

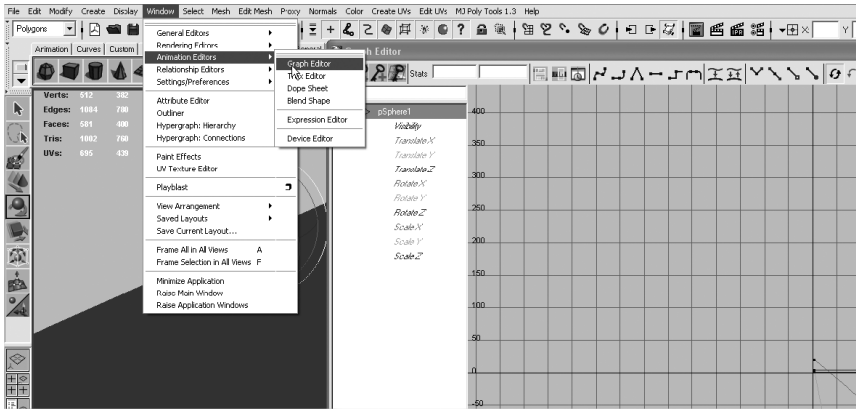


Rysunek 8.8. Ustawianie współrzędnych transformacji w edytorze atrybutów

Przesuwając teraz czarny prostokąt na klatce czasu, możemy zobaczyć, że kula obracając się pokonuje odległość pomiędzy pierwszym, a drugim kluczem. Możemy też nacisnąć przycisk *Play* (po prawej stronie listwy czasu, trójkąt skierowany w prawą stronę), wtedy zobaczymy naszą animację w czasie rzeczywistym. By zatrzymać odtwarzanie, klikamy LKM na ten sam przycisk (który po włączeniu odtwarzania zamienił ikonę na bordowy kwadrat - ikona *Stop*).

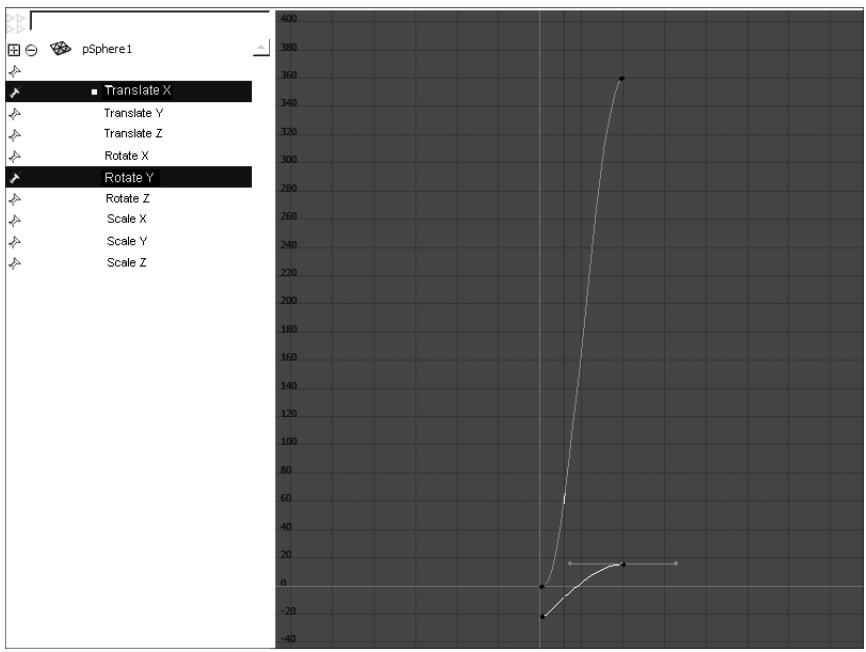
Pierwszym problemem do rozwiązania jest fakt, że kula porusza się ruchem jednostajnym, podczas gdy przy końcu ruchu powinna zwalniać. Do modyfikacji sposobu poruszania się obiektusłuży nam edytor krzywych (*Graph Editor*). Mając zaznaczoną kulę, otwieramy go wybierając polecenie *Window/ Animation Editors/Graph Editor* (Rysunek 8.9).

Na układzie współrzędnych widzimy punkty połączone krzywymi w trzech kolorach. Kolory odpowiadają przesunięciu lub obrotom po osiach X, Y, Z (X



Rysunek 8.9. Otwieranie edytora krzywych

- czerwony, Y - zielony, Z - niebieski). Z lewej strony od układu współrzędnych mamy nazwę modelu kuli (domyślnie *pSphere1*), pod spodem widnieją przypisane do niej atrybuty. Nas interesują jedynie Translate (przesunięcie) i Rotate (obrót). Klikamy LKM na *Translate X* (czyli ruch kuli po osi X, który zaklu-czowaliśmy wcześniej), oraz trzymając wciśnięty CTRL, klikamy LKM na *Ro-*



Rysunek 8.10. Ustawienie krzywej przesunięcia kuli

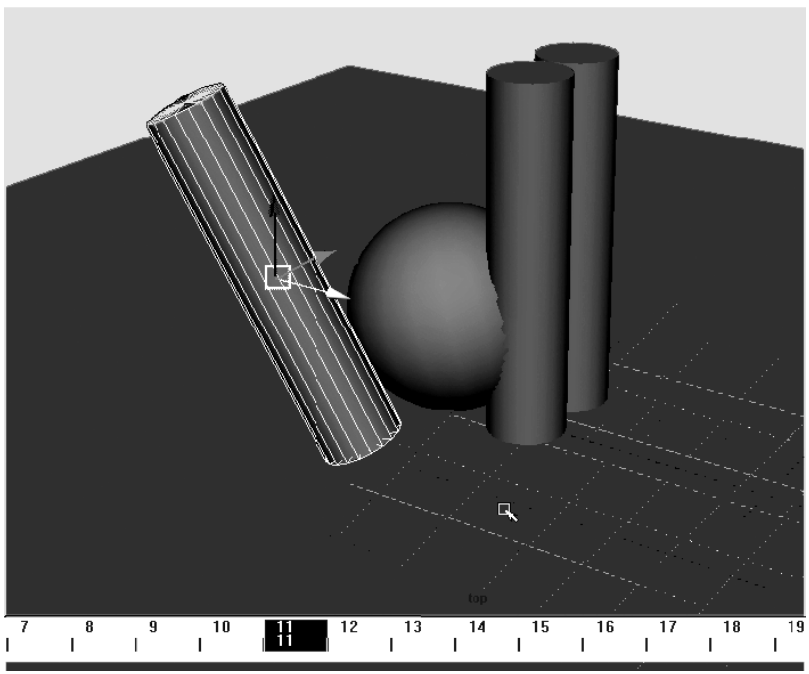
tate Y (czyli nasz obrót kuli po osi Y). Klikamy w drugi klucz (ten po prawej stronie) na krzywej oznaczającej *Translate X* (czerwonej). Zmieni on kolor na żółty, pojawią się także brązowe kreski oznaczające kierunek krzywej, zakończone brązowymi punktami. Klikamy na dowolny z nich i przeciągamy ŚKM w odpowiednią stronę tak, aby krzywa była ustawiona jak pokazuje rys. 8.10.

Identyczną czynność wykonujemy dla zielonej krzywej (oznaczają cej obrót po osi Y), wyginając ją dokładnie w tę samą stronę i w takim samym stopniu co czerwoną krzywą.

Zamykamy edytor krzywych i naciskamy przycisk *Play*. Widzimy teraz, że kula hamuje pod koniec animacji, ruch nie jest jednostajny i wydaje się teraz bardziej realistyczny.

8.3 Animacja upadających kręgli jako obiektów fizycznych

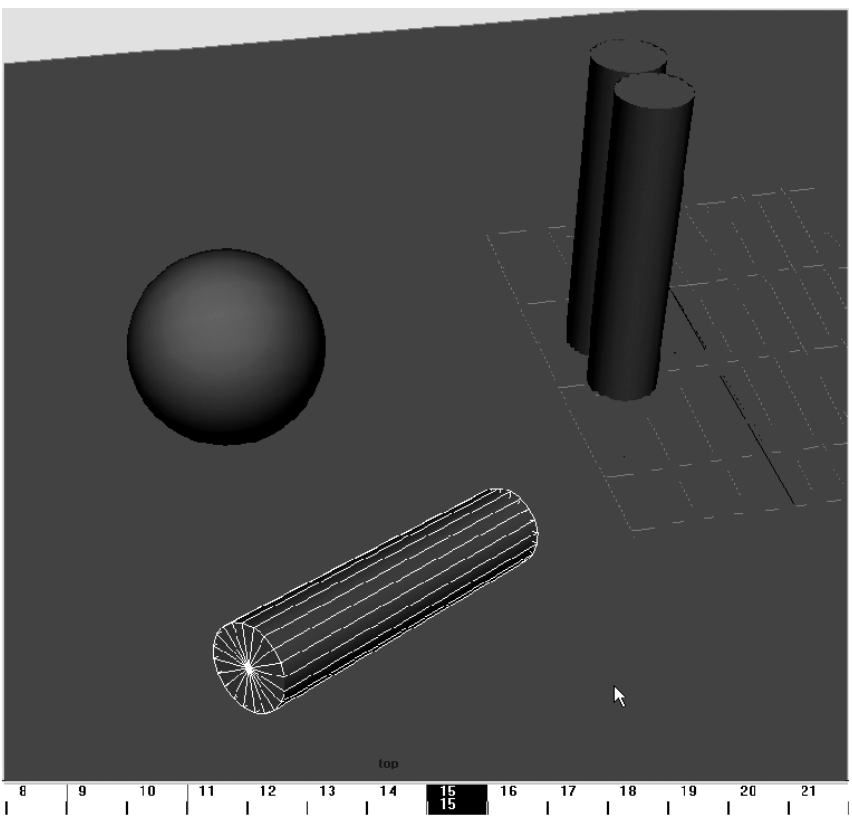
Przejdziemy teraz do trudniejszego zagadnienia - rozbicia kręgli. Wymaga to już nieco więcej wyobraźni i wyczucia ruchu niż toczą ca się kula, gdyż dochodzi tutaj zagadnienie wizualizacji fizyki. Zostanie podany przykład animacji jednego z kręgli, a nabyta w tym rozdziale wiedza powinna wystarczyć do zanimowania pozostałych dwóch.



Rysunek 8.11. Pierwsza faza animacji upadku kręgla

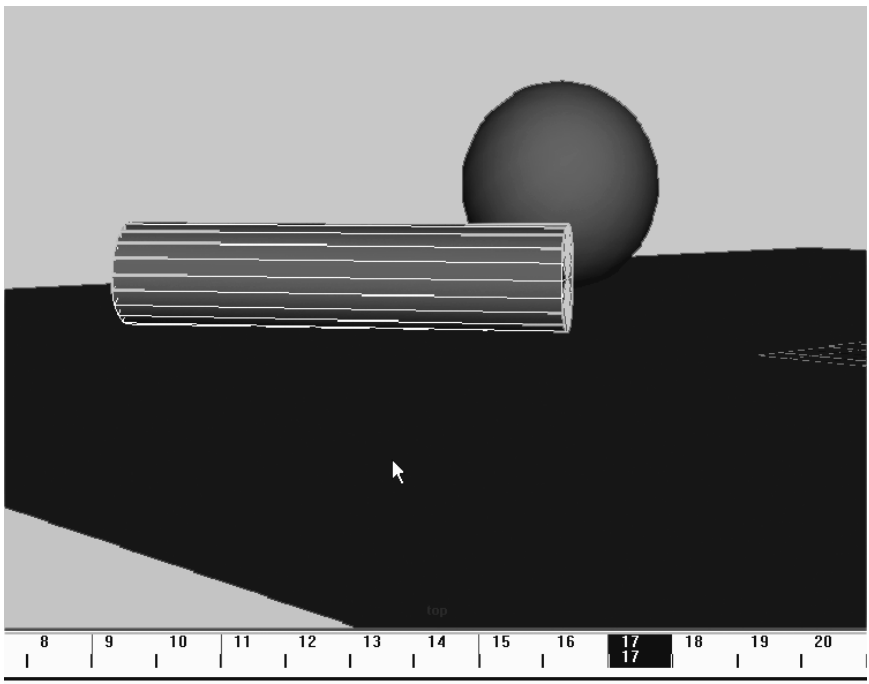
Ustawiamy listwę czasu na klatkę, w której kula styka się z lewym kręglem (ten właśnie będziemy zanimować jako pierwszy). W omawianym przykładzie jest to klatka nr 9. Zaznaczamy kręgiel i stawiamy klucz (S), oznaczając jego początkową pozycję. Dwie klatki dalej (w naszym przypadku to klatka 11.), przesuwamy kręgiel oraz obracamy w lewo, tak by był ustawiony podobnie do przedstawionego na rysunku 8.11.

Następnie, wykonując te same czynności, przestawiamy oraz obracamy kręgiel aby był ustawiony podobnie do tego z rysunku 8.12 i kolejne 2 klatki dalej (u nas w klatce nr 15.) stawiamy następny klucz (kręgiel jeszcze nie powinien leżeć płasko na podłożu - jest to faza tuż przed ostatecznym upadkiem). Ważne jest, aby oglądać wygląd animacji z różnych stron w widoku 3D, żeby oceniać czy wszystko wygląda realistycznie. W każdej chwili można klucz na klatce czasu skasować (przenosząc czarny prostokąt na daną klatkę, klikamy na niej w PKM i wybieramy Delete). Można też klucze przesuwać (SHIFT+LKM na danej klatce listwy czasu zaznaczy ją na czerwono, wtedy przeciągając środkowe czarne strzałki na dole pośrodku zaznaczenia przesuwamy klucz).



Rysunek 8.12. Druga faza animacji upadku kręgla

W trzeciej fazie (na przykład w klatce 17.), kładziemy kręgiel już „na ziemi" (rysunek 8.13), jednak cały czas powinien on się trochę przesuwać w stosunku do wcześniejszej pozycji.

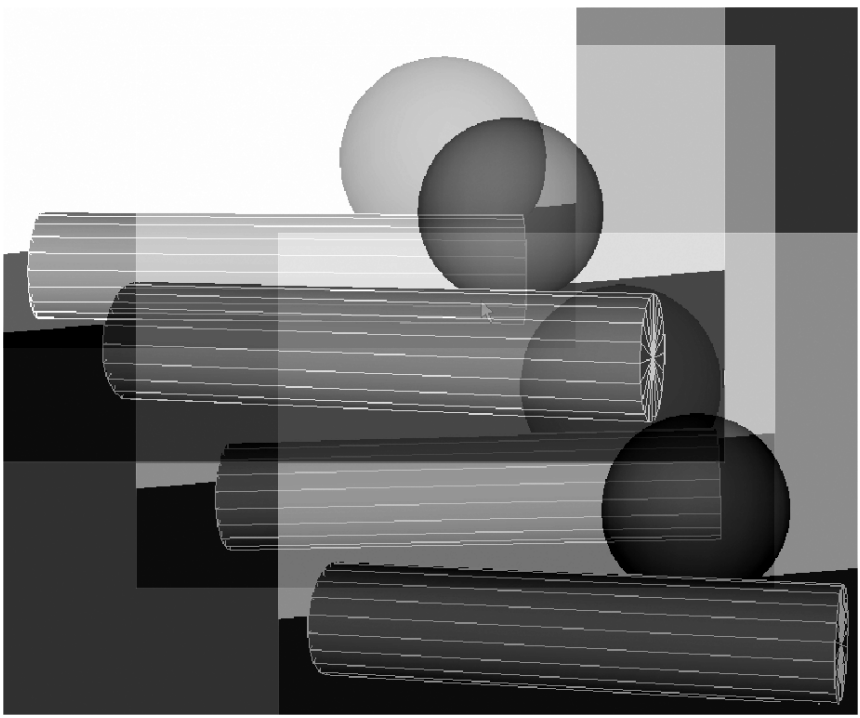


Rysunek 8.13. Trzecia, końcowa faza animacji upadku kręgla

W ostatniej fazie animacji kręgiel powinien się trochę przetoczyć po podłodze i zahamować (oczywiście robimy to edytując krzywe), co doda więcej realizmu naszej animacji.

Po skluczowaniu wszystkich faz ruchu należy pamiętać o kilku rzeczach. Po pierwsze - warto popatrzyć na krzywe ruchu i pamiętać, że swobodnie upadający obiekt porusza się płynnie. To znaczy, że trzeba unikać drastycznych zakrzywień i zmian kierunku ruchu, a linie w edytorze krzywych są tego świetnym zobrazowaniem. Niejednokrotnie dobrze jest też usunąć klucz na danej krzywej (LKM na czarnym punkcie i wciśnięcie klawisza Delete), lub go przestawić (klikając LKM na kluczu i przeciągając go ŚKM). Oszczędza to naprawdę dużo pracy w porównaniu z tym, gdybyśmy chcieli używać tylko kluczowania na listwie czasu. Drugą istotną sprawą jest kształt krzywych w chwili upadku naszego kręgla. Obecnie są one „płaskie", co sprawia wrażenie jakby kręgiel był bardzo ciężki. Lżejsze obiekty w chwili upadku odbijają się zgodnie z fizyką od gruntu. Warto więc trochę „zatrząść" naszym kręglem chwilę po upadku, sta-

wiając nowe klucze w odległości 1-2 klatek od siebie zaraz po zetknięciu z podłożem. Przyjmijmy, że kręgiel upada na swoją lewą stronę, więc chwilę po upadku prawa strona powinna się trochę podnieść (kręgiel należy podnieść odrobinę do góry i przekręcić tak, by prawa strona uniosła się wyżej od lewej, natomiast lewa spoczęła na gruncie), następnie - już w mniejszym stopniu - odwracamy ten ruch, tak, żeby lewa strona się podniosła, a prawa opadła. Wystarczą 2-3 „zatrzęsienia", żeby ruch wydawał się bardziej naturalny. Zobrazowanie tego rozwiązania zostało przedstawione na rysunku 8.14 w postaci nałożenia na siebie kolejnych czterech klatek kończących animację.

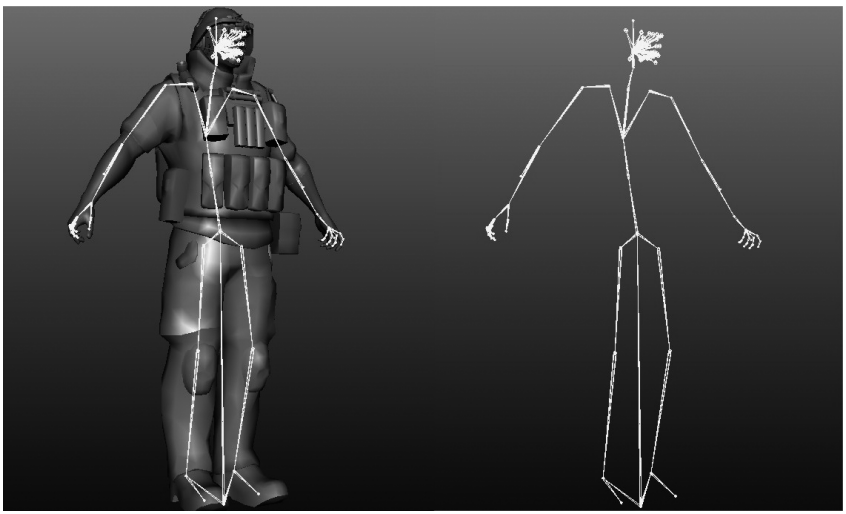


Rysunek 8.14. Fizyka upadku kręgla

8.4 Zasady działania animacji szkieletowej

Animacja szkieletowa służy do symulowania bardziej skomplikowanych ruchów niż proste przesuwanie czy obracanie obiektu w odniesieniu do obiektów o złożonej strukturze. Szkielet, znajdujący się w obiekcie daje nam możliwość wyginania go, zmian kształtu powierzchni itd. Animacje szkieletowe są podstawą poruszania postaci w grach komputerowych czy też filmach 3D. Wszystkie

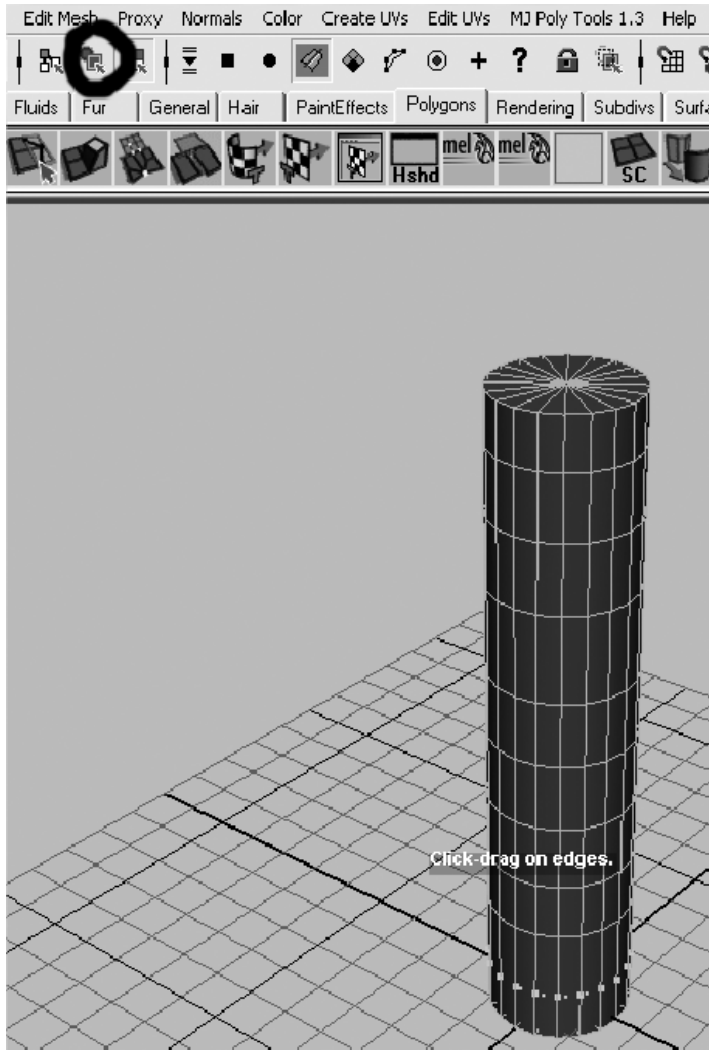
postacie w takich produkcjach mają swoje animowane szkielety, często bardzo skomplikowane. Na rysunku 8.15. przedstawiono przykładowy model postaci do gry 3D wraz z jego szkieletem oraz sam szkielet wykorzystany do animowania tej postaci. W tym podrozdziale nauczymy się tworzyć prosty szkielet oraz go animować na znacznie mniej skomplikowanym przykładzie, stosując jednak te same narzędzia co przy dużo bardziej złożonych obiektach.



Rysunek 8.15. Model postaci do gry first-person-shooter oraz jego szkielet

8.4.1 Przygotowanie prostej sceny pod animację szkieletową

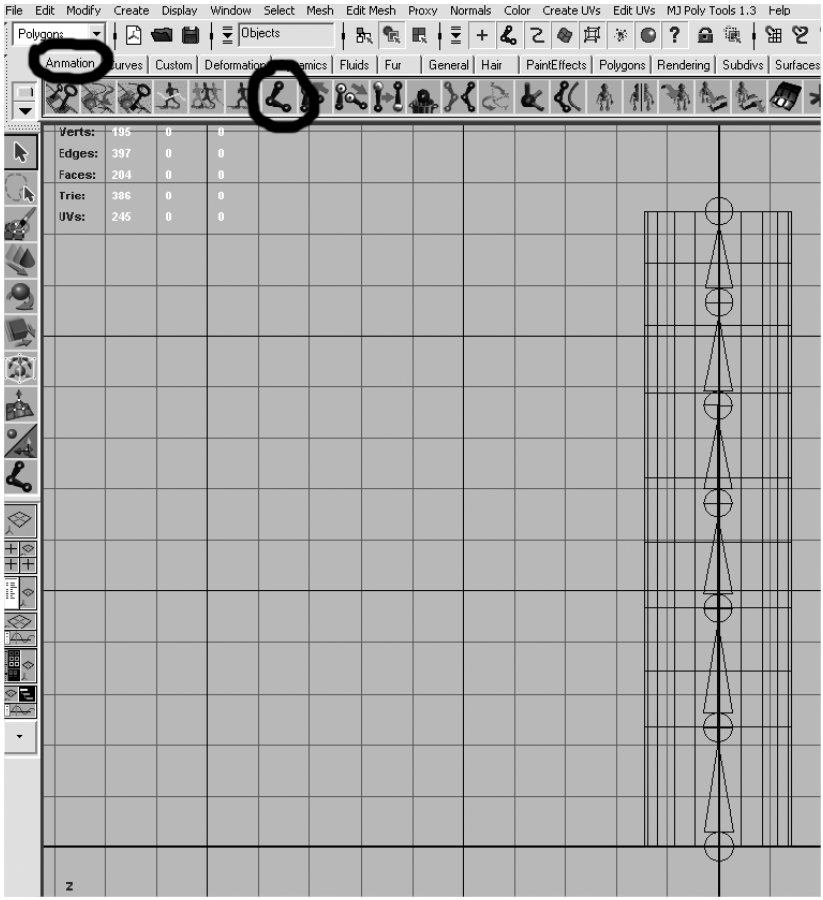
Obiektem, który będziemy animować za pomocą szkieletu będzie prosty cylinder, podzielony w obwodzie na więcej segmentów (co spowoduje, że będzie można stworzyć większą iluzję „miękkości", czy też gładkości powierzchni po jej wygięciu). Tworzymy zatem walec tak jak w poprzednim ćwiczeniu. Następnie wybieramy funkcję *Edit Mesh/Insert Edge Loop Tool*. Klikamy kursorem na którejś z pionowych krawędzi naszego walca, dzieląc go na więcej segmentów. Czynność tę powtarzamy, dotą d, aż efekt będzie podobny do rysunku 8.16 - czyli walec będzie odpowiednio zagęszczony. Następnie wychodzimy z trybu edycji siatki, klikając na ikonkę oznaczoną na rysunku 8.16. Walec można od razu utworzyć z odpowiednio gęstą siatkę poprzez wpisanie odpowiednich parametrów, jednak przywołując tutaj opisaną wyżej metodę zagęszczenia ścianek zwracamy uwagę na to, że dość często w trakcie pracy nad bardziej złożonymi modelami konieczne jest zwiększanie ich szczegółowości w celu uzyskania odpowiednich krzywizn po animacji - zaś w takich przypadkach jednym wyjściem jest ręczne zagęszczanie modelu.



Rysunek 8.16. Użycie narzędzia Insert Edge Loop Tool do zagęszczenia siatki

8.5 Tworzenie prostego szkieletu

Ustawiamy scenę na widok z boku, bez perspektywy po lewej stronie górnej listwy narzędzi klikamy zakładkę *Animation*. Włączamy narzędzie tworzenia kości (*Joint Tool* - siódma ikona od lewej) i klikając LKM w oknie widokowym, w centrum naszego cylindra, tworzymy łańcuch kości podobny do tego z rysunku 8.17 - kolejnymi kliknięciami wstawiamy następne kości, które powinny utworzyć pionowy łańcuch.



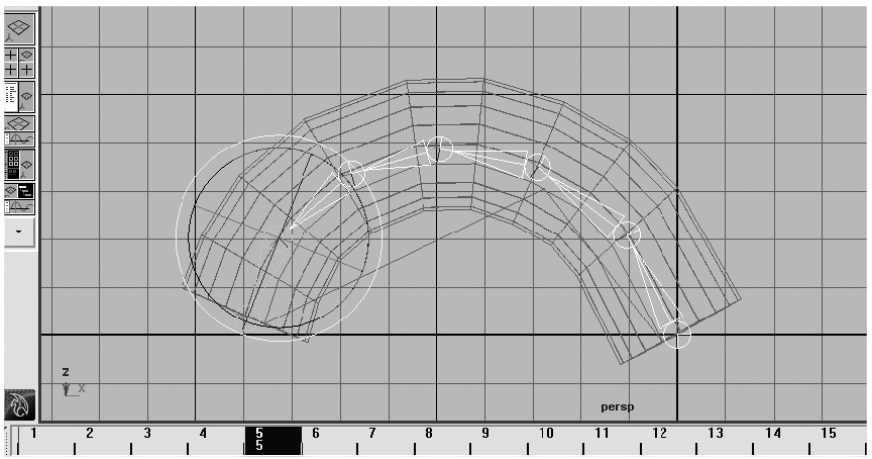
Rysunek 8.17. Tworzenie łańcucha kości

Następnym krokiem jest przypięcie siatki cylindra do szkieletu. Zaznaczamy LKM cylinder, trzymając SHIFT klikamy LKM na dolną kość szkieletu (żeby mieć zaznaczone równocześnie obydwa obiekty - siatkę i szkielet). Na liście po lewej stronie listwy narzędziowej należy mieć teraz włączoną pozycję *Animation*. Z górnej listwy menu wybieramy polecenie *Skin/Bind Skin/Smooth Bind*. Cylinder zmieni kolor na różowy, co oznacza że jest przypięty do szkieletu. Za pomocą narzędzia obracania można zaobserwować w jaki sposób zachowuje się siatka cylindra gdy zaznaczamy i obracamy pojedyncze kości.

8.5.1 Prosta animacja szkieletowa z użyciem osi czasu

Mając aktywną pierwszą klatkę animacji na listwie czasu, zaznaczamy wszystkie kości, trzymając SHIFT i klikając na każdej z nich po kolei LKM. Stawiamy

klucz w pierwszej klatce, naciskając S. Przechodzimy do 5-tej klatki na listwie czasu. Mając ciągle zaznaczone wszystkie kości używamy narzędzia obracania i przekręcamy wszystkie kości naraz w lewo lub prawo (nasz cylinder powinien się gładko wygiąć jak na rysunku 8.18) i stawiamy kolejny klucz (S).



Rysunek 8.18. Wygięcie cylindra

W 15-tej klatce czasu przekręcamy kości tak, by cylinder był pochylony w drugą stronę i kluczujemy tę pozycję, po czy kopiujemy pozycję kości z pierwszej klatki do ostatniej na osi czasu. Naciskając przycisk *Play* możemy pooglądać naszą zapętloną animację, najlepiej z różnych perspektyw. Jest to prosty przykład, dzięki któremu można zrozumieć działanie animacji szkieletowej. Animacja rozbudowanej postaci za pomocą szkieletu jest dużo bardziej skomplikowana, jednak szkoląc się w opisanych tu technikach można zbliżyć się do biegłości w tym zagadnieniu - istotne jest to, że model animowany przedstawionymi tu metodami można bez większego trudu wyeksportować do silnika gry i obejrzeć go tam w czasie rzeczywistym.

Literatura

1. Ernest Adams, Projektowanie gier. Podstawy. Wydanie II, Wydawnictwo Helion, 2010.
2. Adam Bryła, Wojciech Pazdur, Katarzyna Wadas: Grafika komputerowa 3DS MAX, Wydawnictwo PJWSTK 2010.
3. Kamil Bilczyński, Wojciech Pazdur: Modelowanie w programie Maya, Wydawnictwo PJWSTK 2011.
4. David Franson, Andre LaMothe, 2D Artwork and 3D Modeling for Game Artists, Premier Press 2002.
5. Scott Kelby, Photoshop. Efekty specjalne, Wydawnictwo Helion, 2005 Jeannie Novak, Game Development Essentials: An Introduction, Delmar Cengage Learning, 2007.
6. Matthew Omernick, Creating the Art of the Game, New Riders Games, 2004 Anna Owczarz-Dadan, Photoshop CS3 CE. Kurs, Wydawnictwo Helion, 2007.

POLSKO-JAPOŃSKA WYŻSZA SZKOŁA TECHNIK KOMPUTEROWYCH

ポーランド日本情報工科大学

Jedna z najlepszych uczelni w Polsce – wyróżniana przez pracodawców, studentów i media. Od początku swojej działalności zajmuje czołowe miejsce w prestiżowych rankingach uczelni wyższych – wielokrotnie zdobywała pierwsze miejsce w rankingach tygodników „Polityka", „Wprost" i „Newsweek" oraz Perspektyw/ Rzeczpospolitej w kategoriach uczelni technicznych, jak i niepublicznych.

PJWSTK jest uczelnią akademicką – Wydział Informatyki posiada uprawnienia do nadawania stopnia doktora oraz doktora habilitowanego w dziedzinie nauk technicznych.

Uczelnia prowadzi studia na kierunkach:

Architektura Wnętrz – Wydział Sztuki Nowych Mediów
studia I stopnia

Grafika – Wydział Sztuki Nowych Mediów
studia I i II stopnia oraz magisterskie jednolite

Informatyka – Wydział Informatyki
studia I, II i III stopnia oraz studia podyplomowe

Kulturoznawstwo – Wydział Kultury Japonii
studia I i II stopnia

Zarządzanie – Wydział Zarządzania Informacją
studia I stopnia

Przy PJWSTK działają także:
Akademickie Centrum Szkoleniowe
Akademickie Liceum Ogólnokształcące
Niepubliczne Liceum Plastyczne

Główna siedziba znajduje się w samym centrum Warszawy:
ul. Koszykowa 86
02–008 Warszawa
tel.: 22 584 45 00
www.pjwstk.edu.pl
e-mail: pjwstk@pjwstk.edu.pl

Ośrodki w Bytomiu i w Gdańsku dopełniają oferty edukacyjnej:

Wydział Zamiejscowy Informatyki w Bytomiu
Aleja Legionów 2
41–902 Bytom
tel.: 32 387 16 60
www.bytom.pjwstk.edu.pl
e-mail: bytom@pjwstk.edu.pl
kierunki: informatyka, grafika

Wydział Zamiejscowy Informatyki w Gdańsku
Wydział Zamiejscowy Sztuki Nowych Mediów w Gdańsku
ul. Brzegi 55
80–045 Gdańsk
tel.: 58 683 59 75
www.gdansk.pjwstk.edu.pl
e-mail: gdansk@pjwstk.edu.pl
kierunki: informatyka, grafika

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Publikacja współfinansowana ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

WYDAWNICTWO PJWSTK

02–008 Warszawa, ul. Koszykowa 86
tel.: 22 58 44 526, fax: 22 58 44 503
e-mail: oficyna@pjwstk.edu.pl
www.wydawnictwo.pjwstk.edu.pl

ISBN 978–83–63103–15–6

Egzemplarz bezpłatny
Podręczniki akademickie, tom 61