IDŹ DO

PRZYKŁADOWY ROZDZIAŁ

SPIS TREŚCI

KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK

DODAJ DO KOSZYKA

CENNIK I INFORMACJE

ZAMÓW INFORMACJE O NOWOŚCIACH

ZAMÓW CENNIK

CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

Perełki programowania gier. Vademecum profesjonalisty. Tom 2

Autor: Mark DeLoura
Tłumaczenie: Rafał Jońca
ISBN: 83-7197-837-5
Tytuł oryginału: Game Programming Gems 2
Format: B5, stron: 608

Jeśli zajmujesz się lub zamierzasz się zająć programowaniem gier komputerowych, nie odkładaj tej książki na półkę. Znajdziesz w niej siedemdziesiąt rozwiązań rozmaitych problemów, przed którymi staje programista gier. Są to rozwiązania do natychmiastowego zastosowania lub pomysły, które mogą znacznie zredukować nakład pracy. Ich autorami są najwybitniejsi autorzy gier, współtwórcy wielu prawdziwych hitów. Dość powiedzieć, że redaktorami książki „Perełki programowania gier" są pracownicy firm takich jak Nintendo czy NVidia Corporation.

Autorzy postarali się, by przedstawiane przez nich perełki ukazywały praktyczne techniki programistyczne, możliwe do osiągnięcia przy użyciu aktualnie stosowanych technologii i pomagające przy pisaniu gier komputerowych. Przykładowe kody źródłowe opierają się na uznanych standardach: językach C i C++, interfejsy OpenGL i DirectX i nieodzownym, gdy chcemy uzyskać maksymalną wydajność asemblerze procesorów x86.

Porady i rozwiązania podzielone są na 6 części:

- Programowanie ogólne
- Matematyka
- Sztuczna inteligencja
- Zarządzanie geometrią
- Grafika
- Programowanie dźwięku

Dołączony CD-ROM zawiera wszystkie kody źródłowe z książki, dema wielu przedstawionych technik, bibliotekę DirectX 8, instalator glSetup, bibliotekę narzędzi GLUT, obrazy z kolorowej wkładki w wysokiej rozdzielczości.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Podziękowania 15

O obrazku z okładki 16

Biografie Autorów 17

Przedmowa 31

Część I Programowanie ogólne 35

Wprowadzenie 37

Rozdział 1.1 **Optymalizacja kodu w języku C++ w grach** 39
Konstrukcja i destrukcja obiektu 39
Zarządzanie pamięcią 42
Funkcje wirtualne 43
Rozmiar kodu 45
Biblioteka STL 45
Zaawansowane funkcje 47
Dodatkowa lektura 48

Rozdział 1.2 **Rozwijanie funkcji w miejscu wywołania kontra makra** 49
Zalety funkcji inline 49
Kiedy należy używać funkcji inline 51
Kiedy należy używać makr 51
Dodatki w kompilatorze Microsoftu 52
Bibliografia 52

Rozdział 1.3 **Programowanie z wykorzystaniem abstrakcyjnych interfejsów** 53
Abstrakcyjny interfejs 53
Dodawanie fabryki 55
Abstrakcyjne klasy jako zbiory cech 56
Wszystko ma swoją cenę 59
Wnioski 59
Bibliografia 60

Rozdział 1.4 **Eksport klas C++ z bibliotek DLL** 61
Eksport funkcji 61
Eksport klasy 61
Eksport funkcji członkowskich klasy 63
Eksport funkcji wirtualnych klasy 63
Podsumowanie 64

Rozdział 1.5 **Zabezpiecz się przed piekłem bibliotek DLL i brakujących funkcji systemu operacyjnego 65**
Łączenie jawne i niejawne 66
Funkcje LoadLibrary i GetProcAddress 66
Obrona przed błędnymi bibliotekami DirectX 67
Funkcje specyficzne dla systemu operacyjnego 68
Podsumowanie 69

Rozdział 1.6 **Dynamiczna informacja o typie 71**
Wprowadzenie do klasy dynamicznej informacji o typie 71
Demaskowanie typu i odpytywanie DTI 72
Dziedziczenie oznacza „jest typu” 73
Obsługa ogólnych obiektów 74
Implementacja trwałej informacji o typie 75
Zastosowanie trwałej informacji o typie w bazie danych zapisów stanu gry 77
Wnioski 77
Bibliografia 78

Rozdział 1.7 **Klasa właściwości umożliwiająca ogólny dostęp do członków klas C++ 79**
Kod 80
Inne możliwe wykorzystania klas 82
Bibliografia 83

Rozdział 1.8 **Fabryka jednostek w grze 85**
Komponenty 86
Klasy waga piórkowa, zachowań i eksportowa 86
Obiekty wagi piórkowej 86
SAMM-y, gdzie jesteś? 87
Hierarchia klas zachowań 88
Wykorzystanie wzorca szablonu metod w przypisywaniu zachowań 89
Klasy eksportowe 90
Fabryka jednostek 91
Wybór strategii w trakcie działania programu 92
Uwagi końcowe 94
Bibliografia 94

Rozdział 1.9 **Dodawanie klasy informującej o niezalecanych funkcjach w C++ 95**
Możliwe rozwiązania 95
Idealne rozwiązanie 96
Używanie i przypisywanie niezalecanych funkcji 96
Implementacja niezalecania funkcji w C++ 97
Co można poprawić? 98
Podziękowania 98
Bibliografia 98

Rozdział 1.10 **Menedżer pamięci pomocny przy testowaniu gry 99**
Zaczynamy 99
Rejestracja informacji w menedżerze 100
Raportowanie zebranych informacji 102
O czym należy pamiętać 104
Możliwe rozszerzenia 105
Bibliografia 105

Rozdział 1.11 **Moduł profilujący wbudowany w grę 107**
Podstawy profilowania programów 107
Komercyjne narzędzia 108
Czemu utworzyć własny system? 109
Wymagania dotyczące modułu profilującego 109
Architektura i implementacja 110
Szczegóły implementacji 111
Analiza danych 111
Uwagi dotyczące implementacji 112

Rozdział 1.12 **Model programowania liniowego w grach w systemie Windows 113**
Aktualizacja świata 113
Rozwiązanie — wielowątkowość 114
Bibliografia 117

Rozdział 1.13 **Nawijanie stosu 119**
Prosty tymczasowy powrót 119
Łańcuchy tymczasowych powrotów 120
Thunking 122
Rekurencja 123

Rozdział 1.14 **Samomodyfikujący się kod 125**
Zasady kodu RAM 125
Szybkie przekształcanie i kopiowanie bitów 126

Rozdział 1.15 **Zarządzanie plikami z wykorzystaniem plików zasobów 133**
Czym jest plik zasobów? 133
Projekt 134
Implementacja 135
Ostatnie słowa na temat implementacji 137
Wnioski 137
Bibliografia 137

Rozdział 1.16 **Odtwarzanie i rejestrowanie wejść gry 139**
Do czego może się przydać rejestracja wejść? 139
Ile to zajmie? 141
Testowanie rejestracji wejścia 145
Wnioski 145
Bibliografia 145

Rozdział 1.17 **Elastyczny system analizy składniowej tekstu 147**
System analizy składniowej 148
Makra, nagłówki i magia wcześniejszego przetwarzania 148
Wyjaśnienie systemu analizy składniowej 149
Klasa TokenFile 152
Uwagi końcowe 153

Rozdział 1.18 **Interfejs uniwersalnego modyfikatora wartości zmiennych 155**
Analiza wymagań 155
Implementacja 156
Użycie 160
Uwagi 162
Podziękowania 162

Rozdział 1.19 **Generacja rzeczywiście losowych liczb 163**
Pseudolosowość 163
Rzeczywista losowość 164

Źródła losowych wartości 164
Źródła sprzętowe 165
Funkcja mieszająca 165
Ograniczenia 166
Implementacja 166
Jak losowe wartości uzyskujemy z GenRand? 167
Bibliografia 168

Rozdział 1.20 **Wykorzystanie filtrów Blooma do poprawienia wydajności obliczeniowej 169**
Sposób Blooma 169
Możliwe zastosowania 170
Jak to działa? 170
Definicje 170
Pierwszy przykład 171
Drugi przykład 175
Uwagi końcowe 175
Wnioski 176
Bibliografia 176

Rozdział 1.21 **Moduł eksportujący postacie z programu 3D Studio MAX i narzędzia dotyczące animacji 177**
Eksport 178
Bibliografia 187

Rozdział 1.22 **Wykorzystanie kamer internetowych w grach wideo 189**
Inicjalizacja okna przechwytywania danych z kamery 189
Manipulacja danymi z kamery 194
Wnioski 197
Bibliografia 198

Część II Matematyka 199

Wprowadzenie 201

Rozdział 2.1 **Sztuczki z liczbami zmiennoprzecinkowymi — poprawa wydajności w standardzie IEEE 203**
Wprowadzenie 203
Format IEEE liczb zmiennoprzecinkowych 204
Sztuczki liczb zmiennoprzecinkowych 205
Liniowa tablica przeglądowa dla funkcji sinus i kosinus 210
Optymalizacja logarytmiczna pierwiastka kwadratowego 212
Optymalizacja dowolnych funkcji 213
Mierzenie wydajności 216
Wnioski 217
Bibliografia 217

Rozdział 2.2 **Sztuczki z wektorami i płaszczyznami 219**
Wysokość względem płaszczyzny kolizji 219
Szukanie punktu kolizji 220
Odległość do punktu zderzenia 221
Odbicie od płaszczyzny kolizji 222
Zderzenia z tłumieniem 225
Interpolacja w poprzek linii lub płaszczyzny 225

Rozdział 2.3 **Szybkie i elastyczne przecinanie trójwymiarowych linii ... 227**
Co czyni ten algorytm elastycznym? ... 228
Sformułowanie problemu ... 228
Wyprowadzenie równań zwięzłego rozwiązania ... 230
Obsługa odcinków ... 236
Opis implementacji ... 239
Możliwości optymalizacji ... 239
Wnioski ... 240
Bibliografia ... 240

Rozdział 2.4 **Określanie odwrotnej trajektorii ... 241**
Szczególne przypadki ... 243
Optymalizacja implementacji ... 248
Podsumowanie ... 249

Rozdział 2.5 **Ramka transportu równoległego ... 251**
Metoda ... 252
Wnioski ... 255
Bibliografia ... 255

Rozdział 2.6 **Gładkie ścieżki C^2 bazujące na kwaternionach ... 257**
Wprowadzenie ... 257
Interpolacja położenia ... 258
Interpolacja obrotu ... 259
Kierunek obrotu i selektywna negacja ... 260
Interpolacja krzywymi sklejanymi dla kwaternionów ... 261
Osobliwości w wymiernym przekształceniu ... 262
Cięcia kamery ... 262
Kod ... 263
Bibliografia ... 263

Rozdział 2.7 **Rekurencyjne grupowanie wymiarami — szybki algorytm detekcji zderzeń ... 265**
Inne zastosowania ... 266
Słaby punkt algorytmu ... 270
Znajdowanie par będących w kolizji ... 271
Złożoność czasowa ... 273
Wnioski ... 274
Bibliografia ... 274

Rozdział 2.8 **Programowanie fraktali ... 275**
Fraktal plazmowy ... 276
Fraktal błędny ... 276
Fraktal ruchu Browna ... 277
Implementacja ... 278
Wykorzystanie FBM ... 281
Bibliografia ... 282

Część III Sztuczna inteligencja ... 283

Wprowadzenie ... 285

Rozdział 3.1 **Strategie optymalizacji sztucznej inteligencji ... 287**
Strategia 1.: Używaj zachowania sterowanego zdarzeniami, a nie odpytywania ... 287
Strategia 2.: Redukuj niepotrzebne obliczenia ... 288
Strategia 3.: Centralizuj współdziałanie za pomocą zarządców ... 289

Strategia 4.: Rzadziej uruchamiaj sztuczną inteligencję 289
Strategia 5.: Rozłóż przetwarzanie na kilka klatek 290
Strategia 6.: Wprowadź poziomy szczegółowości sztucznej inteligencji 290
Strategia 7.: Rozwiązuj tylko część zagadnienia 291
Strategia 8.: Najcięższą pracę wykonuj poza pętlą czasu rzeczywistego 291
Strategia 9.: Używaj pojawiających się zachowań, by unikać skryptów 292
Strategia 10.: Amortyzuj koszty odpytywań za pomocą ciągłego księgowania 292
Strategia 11.: Jeszcze raz przeanalizuj problem 293
Wnioski 293
Bibliografia 294

Rozdział 3.2 **Mikrowątki sztucznej inteligencji obiektu gry 295**
Prostszy sposób 297
Mikrowątki 297
Zarządzanie stosem 299
Problemy 300
Wniosek 301
Bibliografia 301

Rozdział 3.3 **Zarządzanie sztuczną inteligencją za pomocą mikrowątków 303**
Kawałek po kawałku 303
Dobre zachowanie 304
Wszystko jest w umyśle 305
Problemy 307
Wnioski 309
Bibliografia 310

Rozdział 3.4 **Architektura kolejkowania poleceń w grach RTS 311**
Polecenia gier RTS 311
Kolejkowanie poleceń 312
Polecenia cykliczne 313
Wnioski 316
Bibliografia 316

Rozdział 3.5 **Wysokowydajny system widoczności i wyszukiwania oparty na siatkach 317**
Ogólne przedstawienie zagadnienia 317
Definicje 318
Komponent 1.: Mapy widoczności dla poszczególnych graczy 319
Komponent 2.: Szablony linii widoczności 319
Komponent 3.: Połączona mapa widoczności 321
Usprawnione wyszukiwanie 322
Wnioski 324

Rozdział 3.6 **Mapy wpływu 325**
Mapy wpływu 325
Prosta mapa wpływu 326
Dane komórki mapy wpływu 327
Obliczanie potrzebnych wartości 328
Określanie optymalnego rozmiaru komórki 330
Rozchodzenie się wpływów 330
Branie pod uwagę kształtu terenu 331
Szczególne okoliczności 333
Odświeżanie mapy wpływu 333
Mapy wpływu w trójwymiarowych środowiskach 334
Bibliografia 335

Rozdział 3.7 **Techniki oceny strategicznej** ... **337**
Drzewo przydziału zasobów ... 337
Obliczanie pożądanego przydziału ... 339
Określanie aktualnego przydziału ... 339
Podejmowanie strategicznych decyzji ... 340
Miary wartości ... 341
Graf zależności ... 341
Węzły grafu zależności ... 342
Ekonomiczne planowanie ... 342
Znajdowanie czułych zależności ... 343
Wnioskowanie strategiczne ... 343
Osobowość postaci ... 344
Łączymy wszystko razem ... 345
Bibliografia ... 345

Rozdział 3.8 **Analiza terenu w trójwymiarowych grach akcji** ... **347**
Reprezentacja terenu, która pozwoli go zrozumieć i przeanalizować ... 348
Punkty kontrolne ... 348
Przykładowy teren i potrzeby sztucznej inteligencji ... 349
Analiza taktyczna ... 349
Od wartości taktycznych do właściwości punktów kontrolnych ... 350
Obliczanie właściwości punktów kontrolnych ... 351
Wiedza płynąca z doświadczenia zdobywanego w trakcie gry ... 354
Umieszczanie analizy terenu w grze ... 354
Inne zastosowania ... 355
Wnioski ... 356
Bibliografia ... 356

Rozdział 3.9 **Rozszerzona geometria dla znajdowania drogi metodą punktów widzialności** ... **357**
Definiowanie modelu zderzeń ... 358
Znajdowanie drogi między wielokątami ... 358
Rozszerzaj i zwyciężaj ... 359
Suma Minkowskiego dla wypukłych wielokątów ... 359
Rozszerzanie niewypukłej geometrii ... 361
Dobór kształtu zderzenia ... 362
Wnioski ... 363
Bibliografia ... 363

Rozdział 3.10 **Optymalizacja znajdowania drogi metodą punktów widoczności** ... **365**
Znajdowanie drogi za pomocą punktów widoczności ... 366
Przechowywanie najkrótszej ścieżki do każdego punktu ... 366
Łączenie narożników ... 367
Optymalizacja 2.: Rozważaj tylko ścieżki idące wokół narożników ... 368
Strefy zarysu ... 368
Używanie stref zarysu z podziałem przestrzeni ... 370
Wnioski ... 370
Bibliografia ... 371

Rozdział 3.11 **Stada z zębami: drapieżnicy i ofiary** ... **373**
Całkowicie nowy świat ... 374
Stada z zębami ... 377
Ograniczenia i możliwe rozszerzenia ... 377
Bibliografia ... 378

Rozdział 3.12 **Ogólny automat stanów rozmytych w języku C++ 379**
Dlaczego warto używać FuSM w grach? 380
Jak używać FuSM w grze? 381
Krótkie wprowadzenie do ogólnego automatu stanów skończonych z pierwszego tomu 381
Dostosowywanie oryginalnego automatu do FuSM w C++..........382
Teraz dodaj logikę rozmytą do własnych gier! 383
Bibliografia 383

Rozdział 3.13 **Zmniejszanie eksplozji kombinacji w systemach rozmytych........... 385**
Problem 385
Rozwiązanie 386
Konkretny przykład 388
Konkretny przykład w metodzie Combsa 390
Wnioski 392
Bibliografia 392

Rozdział 3.14 **Używanie sieci neuronowych w grach — konkretny przykład........ 393**
Gra 393
Wielowarstwowy perceptron 394
Dobór wejść 395
Zbieranie danych 396
Trenowanie MLP 397
Wyniki 398
Wnioski 399
Bibliografia 399

Część IV Zarządzanie geometrią 401

Wprowadzenie 403

Rozdział 4.1 **Porównanie metod VIMP 405**
Czynniki 405
„Czysta” metoda VIMP 407
Paski pomijane 410
Wielopoziomowe paski pomijane 411
Tryb mieszany VIMP 413
Tryb mieszany pasków pomijanych 414
Przesuwane okno 414
Podsumowanie 418
Bibliografia 418

Rozdział 4.2 **Upraszczanie terenu za pomocą zazębiania kratek 419**
Zabawa z kratkami 420
Tworzenie mapy 421
Szablony kratek 422
Brzydko, brzydko, brzydko 422
Lepiej, szybciej, silniej 424
Wnioski 424
Bibliografia 425

Rozdział 4.3 **Drzewa kul dla szybkiego określania widoczności, raytrackingu i wyszukiwania zakresowego 427**
Kule otaczające 427
Używanie drzew kul 428
Przykładowa aplikacja 429

Rozdział 4.4 **Skompresowane drzewa sześcianów otaczających równoległych do osi 431**
Krótki przegląd sposobów hierarchicznego sortowania 431
Drzewa AABB 432
Tworzenie drzew AABB 433
Kompresja drzew AABB 433
Aproksymacja wymiarów 434
Wykorzystywanie rekurencji 435
Wydajność w trakcie działania 435
Dalsze rozszerzenia 436
Bibliografia 436

Rozdział 4.5 **Przeszukiwanie drzewa czwórkowego o bezpośrednim dostępie 437**
Gdzie idzie wydajność 438
Usuwanie pośredników 439
Warunki i wymagania 439
Określanie poziomu drzewa 440
Dostosowywanie do sytuacji 442
Określanie położenia 443
Przemieszczanie się po drzewie czwórkowym 444
Dostosowywanie drzewa czwórkowego 444

Rozdział 4.6 **Aproksymacja załamań w akwariach 445**
Obserwacja akwarium 445
Poprawa realizmu 448
Wniosek 448

Rozdział 4.7 **Renderowanie zrzutów ekranowych o rozdzielczości nadającej się do druku 449**
Podstawowy algorytm 450
Uwagi i ewentualne problemy 452
Zakończenie 452
Bibliografia 452

Rozdział 4.8 **Stosowanie śladów dla dowolnych powierzchni 453**
Algorytm 453
Przycinanie trójkątów 455
Implementacja 456
Bibliografia 456

Rozdział 4.9 **Rendering odległej scenerii jako tło 459**
Podstawowa metoda 459
Rozdzielczość tła 459
Rozmiar sześcianu tła 460
Rendering sceny 461
Sześcienne mapowanie środowiska 461
Tworzenie tekstur tła 462
Wniosek 462
Kod źródłowy 462

Rozdział 4.10 **Postacie rzucające na siebie cień 463**
Wcześniejsze metody 463
Podział geometrii postaci 463
Rendering tekstury 464
Rendering postaci 464
Wniosek 465
Bibliografia 465

Rozdział 4.11 **Sterowanie i animacja z pozycji trzeciej osoby w grze Super Mario 64 .... 467**
Ustawienia 467
Konwersja wejścia kontrolera 468
Obrót postaci 469
Przesuwanie postaci 470
Animacja postaci 471
Analiza animacji Super Mario 64 473
Wniosek 474
Bibliografia 474

Część V Wyświetlanie grafiki 475

Wprowadzenie 477

Rozdział 5.1 **Rendering kreskówek — wykrywanie i rendering krawędzi sylwetki w czasie rzeczywistym 479**
Twórca zarysów 479
Ważne krawędzie 480
Metody wykrywania krawędzi sylwetki 481
Wyciąganie rysunku w tuszu bazujące na krawędziach 481
Wyciąganie rysunku w tuszu za pomocą programowanego shadera wierzchołków 483
Wyciąganie rysunku w tuszu za pomocą zaawansowanych funkcji tekstur 485
Wniosek 486
Bibliografia 486

Rozdział 5.2 **Rendering kreskówek za pomocą mapowania tekstur i programowanych shaderów wierzchołków 487**
Cieniowanie w kreskówkach 487
Malowanie 488
Programowane shadery wierzchołków 491
Wniosek 493
Bibliografia 493

Rozdział 5.3 **Metody dynamicznego oświetlenia dla pikseli 495**
Trójwymiarowe tekstury w dynamicznym oświetlaniu 495
Mapowanie nierówności dot3 498
Normalizacja za pomocą map sześciennych 501
Światła stożkowe bazujące na pikselach 502
Bibliografia 503

Rozdział 5.4 **Tworzenie proceduralnych chmur za pomocą kart graficznych z akceleracją 3D 505**
Właściwości chmur 505
Generator liczb losowych 506
Animacja oktawy szumu 508
Mapowanie na geometrię nieba 511
Dodatkowe funkcje 511
Ograniczenia sprzętowe 512
Uzyskiwanie większej uniwersalności 513
Wnioski 513
Bibliografia 514

Rozdział 5.5 **Maskowanie tekstur w celu przyspieszenia rysowania efektu soczewki 515**
Zasłanianie efektu soczewki 515
Problemy sprzętowe 516
Maskowanie tekstur 518
Uwagi co do wydajności 519
Usprawnienia 520
Przykładowy kod 520
Alternatywne podejścia 521
Bibliografia 521

Rozdział 5.6 **Praktyczne cienie oparte na buforach priorytetowych 523**
Porównanie buforów priorytetowych z buforami głębi 525
Rozwiązywanie problemów z aliasingiem 526
Metody hybrydowe 528
Podsumowanie 529
Bibliografia 529

Rozdział 5.7 **Oszuści — dodawanie śmieci 531**
Cały proces 532
Rendering oszusta 532
Heurystyka aktualizacji 536
Wydajność 538
Przewidywanie 538
Podsumowanie 539

Rozdział 5.8 **Działania w sprzęcie pozwalające na proceduralną animację tekstur 541**
Operacje sprzętowe 542
Dalsza praca 551
Podziękowania 552
Przykładowy kod 552
Bibliografia 552

Część VI Programowanie dźwięku 553

Wprowadzenie 555

Rozdział 6.1 **Wzorce projektowe w programowaniu dźwięku w grach 557**
Most 557
Fasada 558
Złożenie 559
Pośrednik 559
Dekorator 560
Polecenie 560
Pamiątka 561
Obserwator 561
Wielka kula błota (zwana także kodem spaghetti) 562
Wniosek 563
Bibliografia 563

Rozdział 6.2 **Metoda natychmiastowego ponownego użycia głosów w syntezatorze bazującym na próbkach 565**
Problem 565
Pomysł na rozwiązanie 566
Rozwiązanie 567
Wniosek 568

Rozdział 6.3 **Programowe efekty DSP** **569**
Filtrowanie 569
Splot 570
Opóźnienie 571
Interpolacja 572
Bibliografia 572

Rozdział 6.4 **Interaktywny potok przetwarzania w cyfrowym dźwięku** **575**
Wprowadzenie 575
Dyskusja 578
Kod 580
Dodatkowy komentarz 583
Wniosek 584

Rozdział 6.5 **Prosty sekwenser muzyki dla gier** **585**
Strumieniowanie kontra sekwencjonowanie 585
Podstawowe koncepcje komputerowej muzyki 586
Implementacja komputerowego sekwensera muzyki 589
Sterowanie syntezą dźwięku 596
Kod 596
Wnioski 596
Bibliografia 596

Rozdział 6.6 **Sekwenser interaktywnej muzyki dla gier** **597**
Powiązania muzyczne 597
Znaczenie muzyki 598
Przejścia 598
Rodzaje przejść 598
Czułość sterowania 601
Sterowanie docelowe 601
Przykłady projektów 602
Kod 604
Wniosek 604
Bibliografia 605

Rozdział 6.7 **Interfejs programistyczny niskiego poziomu dla dźwięku** **607**
Podstawowe klasy 607

Dodatki **609**

Skorowidz **611**

Rozdział 5.4

Tworzenie proceduralnych chmur za pomocą kart graficznych z akceleracją 3D

Kim Pallister, Intel
kim.pallister@intel.com

W wielu grach akcja rozgrywa się na zewnątrz budynków, a środowisko przypomina Ziemię. Z tego powodu realistyczny rendering terenu stał się jakby świętym Graalem dla wielu twórców gier. Niestety, na modelowanie nieba i chmur nie zwraca się tyle uwagi, ile się im należy. Chmury to przeważnie jedna lub dwie warstwy statycznych, przesuwanych tekstur. Na pierwszy rzut oka wszystko jest w porządku, ale po dłuższej chwili zauważamy ich powtarzalność, a po dniu grania zacznie nas ona nudzić.

W tym rozdziale zajmiemy się proceduralnie tworzonymi teksturami chmur, które posiadają pewne właściwości rzeczywistych chmur. Ponieważ tekstury przeważnie znajdują się w pamięci podsystemu graficznego, będziemy chcieli je generować prawie wyłącznie za pomocą procesora graficznego. Dodatkowo omówimy kilka modyfikacji techniki pod względem zmiany jakości i wydajności, by dostosować ją do różnych systemów.

Właściwości chmur

Dokładnie przyglądając się charakterystyce rzeczywistych chmur, stosunkowo łatwo można utworzyć listę posiadanych przez nie cech. Oczywiście, jak w każdej metodzie czasu rzeczywistego, będziemy musieli poświęcić kilka z tych funkcji na rzecz szybkości, ale tym będziemy się przejmować później.

Oto kilka spraw, które można zauważyć od razu:

- Chmury są animowane. Po niebie przesuwa je wiatr, ale to nie wszystko: ich kształt zmienia się lub „ewoluuje” w czasie (można to łatwo zauważyć na dowolnych zdjęciach wykonanych w stałych odstępach czasu). Co więcej, liczba chmur na niebie cały czas się zmienia. To rozważanie prowadzi nad do trzech zmiennych: szybkość symulacji, szybkość wiatru i dodatkowego elementu opisującego zachmurzenie.
- Mniejsze elementy zmieniają się szybciej od większych (dotyczy zmiany kształtu).
- Chmury wykazują ogromne samopodobieństwo w swojej strukturze.
- Poziom zachmurzenia zmienia się od pełnego do czystego nieba z tylko kilkoma odosobnionymi chmurkami (lub całkowitym ich brakiem). Gdy zmienia się zachmurzenie, wygląd chmur także ulega zmianie. Dla zachmurzenia chmury stają się szare i ciemne, a dla pogodnego nieba są białe z przebłyskami niebieskiego nieba.
- Chmury to trójwymiarowe istoty. Są oświetlane przez słońce z jednej strony, a cień rzucają z drugiej. W mniejszej skali jest jeszcze trudniej, ponieważ chmury rzucają cienie na inne chmury i odbijają promienie świetlne we wszystkich kierunkach.
- Chmury wyglądają całkowicie inaczej przy wschodzie i zachodzie słońca, ponieważ słońce oświetla je z boku, a w skrajnych przypadkach nawet z dołu.
- Chmury mają tendencję do pływania na wspólnej wysokości, formując warstwy. Czasem tych warstw jest kilka. Ponieważ warstwy te zachowują stałą wysokość nad krzywizną Ziemi, wykorzystamy tę krzywiznę w obliczaniu warstw chmur.

... a to tylko część spostrzeżeń obserwatora patrzącego z Ziemi. W grach takich jak symulator lotu, gdzie możemy wlatywać w chmury, pojawia się wiele dodatkowych utrudnień. W tym rozdziale pozostaniemy na Ziemi.

Generator liczb losowych

Jak w przypadku prawie każdej proceduralnej tekstury, musimy zacząć od określenia generatora pewnego rodzaju szumu. Szum to termin używany do określenia czegoś, co jest z natury losowe. Szumem może być na przykład funkcja, która po podaniu szeregu licz całkowitych (czyli 1, 2, 3...) utworzy pozornie losowy szereg wyników (na przykład 0,52, –0,13, –0,38...). Pozornie losowy, ponieważ dla takiego samego wejścia wyjściowe liczby zawsze będą takie same. Takie funkcje nazywamy pseudolosowymi, ponieważ stosując to same ziarno, możemy odtworzyć te same wyniki. Dodatkowo szum ma często określony wymiary (na przykład dwu- lub trójwymiarowy szum). Odnosi się do liczby wejść przekształcanych na losową wartość — tworzymy po prostu wielowymiarową tablicę. Jeden z wymiarów jest skalowany przez pewien stosunkowo duży czynnik (przeważnie liczbę pierwszą), aby zminimalizować możliwość powtarzania się ciągów liczb.

Tworzenie losowych (lub pseudolosowych) liczb to bardzo rozległy temat. Każde podejście to równowaga między złożonością funkcji a jakością wyników. Dobre generatory liczb losowych dobrze rozkładają losowe wartości po całym przedziale i nie powtarzają się przez bardzo długi czas.

Na szczęście na potrzeby tego rozdziału wystarczy bardzo prosty generator liczb losowych. Gdy poznasz dokładnie tę technikę, dowiesz się, że generator liczb losowych będziemy wywoływali kilka razy na piksel z różnymi wartościami ziarna. Połączenie wyników zamaskuje pojawiającą się powtarzalność.

Generator liczb pseudolosowych, od którego zaczniemy, przedstawia wydruk 5.4.1. Wykorzystuje on liczbę podaną jako parametr (x) w wielomianie, którego współczynniki są mnożone przez duże liczby losowe. W ten sposób uzyskujemy znacznie mniejszą powtarzalność wartości. Maskujemy bit znaku, dzielimy liczbę do zakresu 0 – 2, a następnie odejmujemy ją od jedynki, by otrzymać przedział od –1 do 1.

Wydruk 5.4.1. *Prosty generator liczb pseudolosowych*

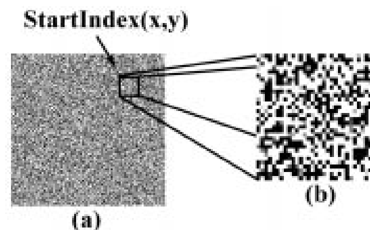
```
float PRNG( int x)
{
  x = (x<<13)^x;

  int Prime1 = 15731;
  int Prime2 = 789221;
  int Prime3 = 1376312589;

  return (1.0f - ((x * (x * x Prime1 + Prime2) + Prime3) & 7fffffff) / 1073741824.0)
}
```

W przypadkach, w których używamy kilku oktaw, możesz zwiększyć „losowość" generatora, tworząc tablicę liczb pierwszych i używając różnych wartości `Prime1` i `Prime2` w zależności od oktawy. Dla naszych celów wykorzystanie tylko jednego zestawu liczb pierwszych jest wystarczające.

Ponieważ chcemy zaimplementować generator w procesorze graficznym, który nie pozwala na wykonanie całego kodu z wydruku 5.4.1, utworzymy tablicę przeglądową tej funkcji jako mapę tekstury 512×512. Wykorzystanie takiej mapy daje nam około 260 tysięcy wpisów, zanim funkcja zacznie się powtarzać. Wykorzystamy tę teksturę jako generator liczb losowych, kopiując fragmenty tej tekstury do docelowej tekstury i używając losowego przesunięcia współrzędnych tekstury wygenerowanego przez oprogramowanie. Przedstawia to rysunek 5.4.1. Kopiowanie wykorzystuje kartę graficzną, aby zrenderować na wielokącie losową teksturę ze źródłowej.

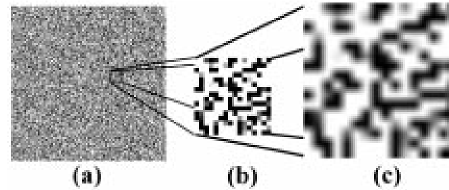


Rysunek 5.4.1.
a) Tekstura stanowiąca tablicę przeglądową losowych liczb;
b) Wygenerowana tekstura szumu 32×32

Szum o ograniczonej zmianie

Ken Perlin po raz pierwszy zastosował szum o ograniczonej zmianie jako metodę renderingu. Szum o ograniczonej zmianie jest ograniczony do pewnego zakresu częstotliwości, który można rozumieć jako maksymalną zmianę między próbkami. W naszym przypadku

oznacza to możliwość utworzenia losowych liczb z maksymalną zmianą miedzy nimi, a co za tym idzie, gładką interpolację losowych próbek. Do interpolacji możemy wykorzystać filtrowanie dwuliniowe karty graficznej. W ten sposób pozbędziemy się składowych o wysokich częstotliwościach i uzyskamy gładkie, bardziej naturalne przejścia. Patrz rysunek 5.4.2.



Rysunek 5.4.2.
a) Próbkowanie tekstury stanowiącej tablicę przeglądową losowych liczb w celu utworzenia tablicy szumu. b), c) Używając zwiększania próbkowania i filtrowania tworzymy szum o ograniczonej zmianie w docelowej rozdzielczości (docelowym zakresie częstotliwości)

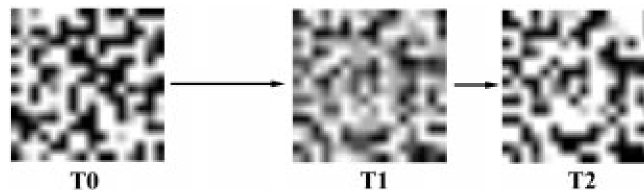
Warto wspomnieć, że w pewnych przypadkach, wykorzystując szum do tworzenia proceduralnej zawartości, interpolacja dwuliniowa nie jest odpowiednia i trzeba skorzystać z lepszej metody filtrowania, by osiągnąć pożądany wynik. Na szczęście w tym przypadku niska jakość filtrowania dwuliniowego jest wystarczająca.

Utworzona w ten sposób tablica szumu daje nam pojedynczy podstawowy szum o danej częstotliwości. Będziemy go nazywać oktawą szumu, ponieważ w dalszej części rozdziału połączymy kilka oktaw (mnożenia o tej samej częstotliwości) razem. Najpierw jednak będziemy musieli zająć się animacją tablicy szumu.

Animacja oktawy szumu

Jeśli chcemy animować oktawę szumu, pomyślmy o czasie jak o trzecim wymiarze, za pomocą jakiego możemy indeksować generator liczb losowych. Możemy to wykonać na karcie graficznej, co jakiś czas zapamiętując aktualny szum, tworząc nowy i interpolując między nimi od jednego uaktualnienia do drugiego. Szybkość uaktualnień tekstury określa częstotliwość w trzecim wymiarze.

Interpolację przedstawia rysunek 5.4.3. Jest to jedyny przypadek, gdzie wykorzystanie interpolacji liniowej nie jest idealne, ponieważ szum jest bardziej „skupiony" w rzeczywistych punktach próbkowania. Jak się jednak za chwilę przekonasz, problem ten nie jest widoczny w końcowym efekcie, ale warto o nim pamiętać. Jeśli chcemy poświęcić wydajność rysowania, możemy obliczać oktawy dla wielu punktów w czasie i używać lepszej interpolacji w celu uzyskania lepszych wyników.



Rysunek 5.4.3.
*Interpolacja między uaktualnieniami szumu w celu animacji oktawy. Wynik = WcześniejszaAktualizacjia * (1 – (T1 – T0)/(T2 – T0)) + NowaAktualizacja * (T1 – T0)/(T2 – T0). T0 to czas wcześniejszej aktualizacji, T2 to czas nowej aktualizacji, a T1 to aktualny czas między aktualizacjami*

Sumowanie oktaw w celu utrzymania burzliwego szumu

Popularną metodą używaną przy tworzeniu proceduralnych tekstur jest sumowanie fraktalne, które polega na sumowaniu skalowanych harmonicznych podstawowej funkcji szumu. Z takiej operacji powstaje fraktal ruchu Browna, który jest bardzo popularny w wielu technikach renderingu, na przykład fraktalnego terenu i wielu typów proceduralnych tekstur. Sumę fraktalną przedstawia równanie 5.4.1. *a* określa sposób przechodzenia przez spektrum częstotliwości (*a* ma przeważnie wartość 2, czyli utrzymujemy f, 2f, 4f, ...), a *b* określa amplitudę dodawanego elementu z szeregu.

$$Szum(x) = \sum_{k=0}^{N-1} \frac{Szum(a^k x)}{b^k} \tag{5.4.1}$$

Na szczęście wartość 2 dla *a* jest zarówno powszechnie używana w generatorach proceduralnych tekstur, jak i dobrze obsługiwana przez sprzęt, ponieważ możemy po prostu zaimplementować oktawy szumu jako szereg map tekstur o rozmiarach będących kolejnymi potęgami 2.

Wartość 2 dla *b* także jest powszechnie używana i upraszcza naszą implementację. Złożenie oktaw szumu można wykonać, przeprowadzając kilka przebiegów renderingu z prostym mieszaniem (patrz pseudokod z wydruku 5.4.2).

Wydruk 5.4.2. *Złożenie oktaw szumu*

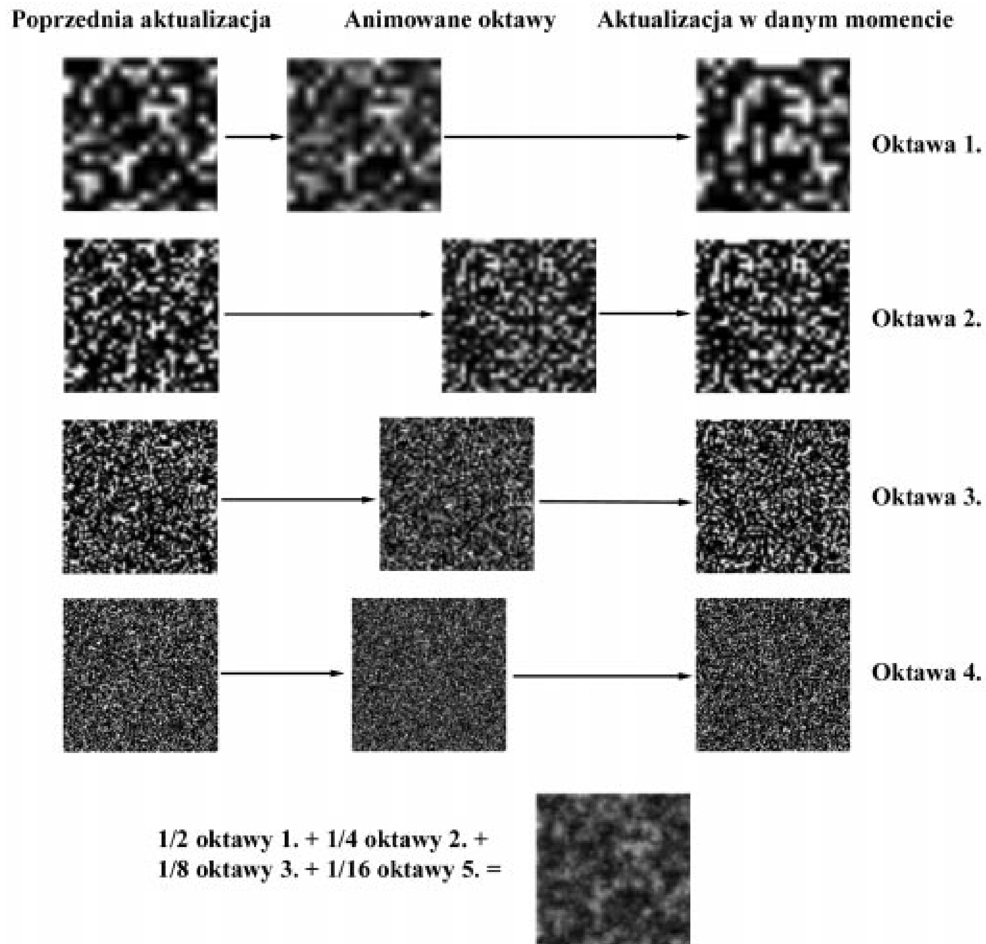
```
// Zauważ, że ScaleFactor ma za zadanie utrzymać wyniki w przedziale 0 - 1.
// Zmienia więc {1 + 1/2 + 1/4 + ...} na {1/2 + 1/4 + ...}
float ScaleFactor = 0.5
float FractalSum = 0.0
for i = 1 to NumOctaves
{
  FractalSum = FractalSum*0.5 + ScaleFactor * NoiseOctave(NumOctaves-i)
}
```

Rysunek 5.4.4 przedstawia proces animacji różnych oktaw między aktualizacjami i łączenia ich w jedną teksturę burzliwego szumu.

Tworzenie chmur z pary

Mamy teraz mapę tekstury z animowanym szumem z turbulencjami i musimy wykonać kilka kroków, by przekształcić ją na chmury. Idealnie byłoby wykorzystać szum jako wejście pewnego rodzaju funkcji wykładniczej, uzyskując ostry „poziom skroplenia”, powyżej którego chmury są widoczne. Ponieważ jednak nie mamy dostępu do takiej operacji w procesorze grafiki, skorzystamy z innej metody. Istnieje kilka sposobów poradzenia sobie z sytuacją.

Najprostsze podejście to odejmowanie. Odejmujemy określoną wartość od tekstury i przycinamy wynik od dołu do 0. W ten sposób uzyskujemy pojedyncze chmury (patrz rysunek 5.4.5).



Rysunek 5.4.4. *Połączenie kilku oktaw animowanego szumu*

Rysunek 5.4.5. *Odejmowanie stałej wartości w celu otrzymania pojedynczych chmur*

Niestety, w ten sposób tracimy część dynamicznego zakresu dla chmur, które pozostały. Możemy to skompensować, zwiększając nasycenie i mnożąc kolory wierzchołków. Tego sposobu używamy w przykładowym kodzie.

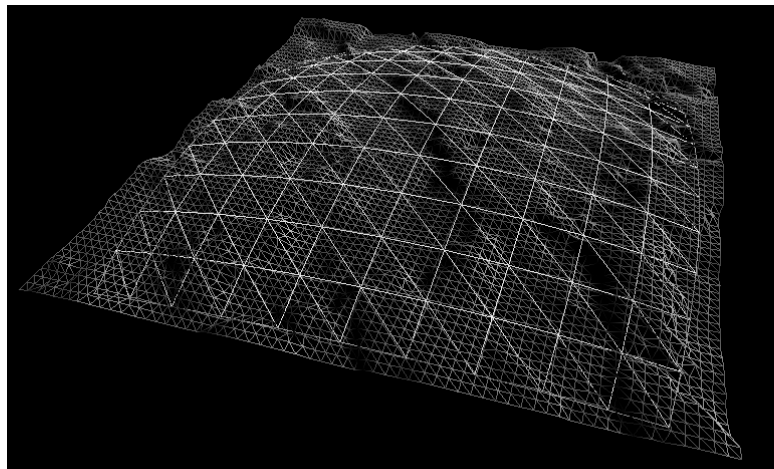
Inna metoda polega na dodaniu kanału krycia do wszystkich omawianych do tej pory warstw tekstur. Wykonamy tę samą operację odejmowania co wcześniej, ale na kanale krycia, a następnie wykorzystamy test krycia do zamaskowania niektórych regionów bez tracenia dynamicznego zakresu. W tej metodzie pojawia się jednak inny rodzaj problemu. Ponieważ test krycia jest wykonywany przed filtrowaniem, pojawią się ostre krawędzie, jeśli nie skorzystamy z tekstury o naprawdę wysokiej rozdzielczości.

Jeszcze jedną metodą jest pewnego rodzaju przeglądanie tekstury, gdzie kolor tekstury wpływa na położenie, w którym teksel jest umieszczany w kolejnym etapie. Przykładem takiego zachowania może być tryb renderingu `BumpEnvMap` obsługiwany przez niektóre karty w bibliotece DirectX. W przyszłości, gdy więcej kart będzie obsługiwało ten tryb, stanie się możliwe zakodowanie wykładniczej funkcji w mapie tekstury i przeglądanie jej w celu otrzymania wyniku.

Mapowanie na geometrię nieba

Po przygotowaniu tekstury chmury z rysunku 5.4.5 możemy ją mapować na geometrię nieba. Wybór geometrii zależy od aplikacji. W przykładowym programie używamy tzw. płaszczyzny nieba, która jest po prostu prostokątną siatką trójkątów z wierzchołkami przesuniętymi tak, że stanowią wycinek kuli. Wyobraź to sobie jak rzucenie skrawka płachty na powierzchnię piłki plażowej. Nie jest to idealne rozwiązanie, ale mapowanie tekstury jest proste. Płaszczyznę nieba przedstawia rysunek 5.4.6.

Rysunek 5.4.6.
Geometria terenu i płaszczyzna nieba używana dla tekstury nieba



Dodatkowe funkcje

W tym momencie możemy wykonać wiele rzeczy z chmurami, w zależności od tego, jak wiele czasu procesora i karty graficznej możemy na to poświęcić. Kilka z tych propozycji zostało zaimplementowanych w przykładowym kodzie; pozostałe to propozycje dalszych usprawnień dla Twoich implementacji. Oto kilka funkcji, które możesz dodać:

- Sterowany szumem kierunek wiatru i zachmurzenie. Pewne interesujące wyniki można uzyskać, korzystając z innych funkcji szumu do modyfikacji pozostałych zmiennych w czasie, więc na przykład wiatr może zmieniać kierunek, a poziom chmur obniżać się i podwyższać w ciągu kilku godzin lub dni.
- Wytłaczać chmury, dodając wrażenie trójwymiarowości. Wymaga to jednak dodatkowego przebiegu i pewnych modyfikacji w wartościach UV wierzchołków. Wierzchołkom dodajemy drugi zbiór współrzędnych tekstury przesuwanych w zależności od kierunku słońca. Chmury są ciemniejsze po stronie przeciwnej do słońca, a jaśniejsze po tej samej.

- Chmury mogą rzucać cień na ziemię. Wystarczy użyć wynikowej tekstury z mieszaniem odejmującym. Teren musi posiadać inny zestaw współrzędnych tekstury lub używać rzutowania tekstury.
- Modyfikować oświetlenie i (lub) intensywność efektu soczewki (*lens flare*). Ponieważ wiemy, jak tekstura jest mapowana na geometrię i znamy kąt nachylenia słońca, możemy policzyć dokładnie teksel lub grupę tekseli, które znajdują się na linii widoczności względem słońca. Możemy w ten sposób zmniejszać oświetlenie terenu lub czasowo wyłączać efekt soczewki. Zauważ, że modyfikacja oświetlenia spowodowana większym zachmurzeniem zmniejsza intensywność światła kierunkowego słońca, ale zwiększa oświetlenie ogólne, ponieważ chmury rozpraszają promienie słońca we wszystkich kierunkach.

Ograniczenia sprzętowe

Pozwolenie karcie graficznej na przeprowadzenie większości operacji w tej technice umożliwia osiągnięcie większej wydajności niż w przypadku, w którym musielibyśmy ręcznie blokować i modyfikować teksturę w każdej klatce. Niestety, takie podejście ma także wady. Oto one:

- **Obsługa renderingu do tekstury**. Metoda przedstawiona w tym rozdziale intensywnie korzysta z tej funkcji. Niestety, obsługa tej funkcji nie występuje we wszystkich kartach graficznych. Wykrywanie obsługi staje się łatwiejsze przy nowoczesnych interfejsach programistycznych, ale nadal nie jest pewne. W przypadkach, gdy rendering do tekstury nie jest obsługiwany, można skorzystać z kopiowania bufora ramki do tekstury, ale prawdopodobnie ucierpi na tym wydajność.
- **Ograniczona precyzja**. Aktualnie istniejący sprzęt graficzny przechowuje tekstury i renderuje docelowe wartości jako liczby całkowite, przeważnie 16 lub 32 bity na piksel. Ponieważ w zasadzie działamy na skali szarości, jesteśmy ograniczeni do 8 bitów na piksel, a w najgorszym przypadku do 4! Ta druga wartość jest bardzo mała, więc na pewno pojawią się błędy. Zauważ, że oznacza to, że oktawy o wysokiej częstotliwości z tego powodu dodają tylko kilka bitów do końcowego wyniku.
- **Ograniczony dynamiczny zakres**. Ponieważ liczby całkowite muszą reprezentować dane z zakresu 0 – 1, jesteśmy zmuszeni wcześniej przeskalować i przesunąć te wartości do odpowiedniego zakresu. Ponieważ nasza funkcja szumu zwraca wartości w przedziale od –1 do 1, musimy je zmodyfikować, by uzyskać pożądany zakres. Jest to dodatkowa praca, która powiększa błędy powodowane przez ograniczoną precyzję.
- **Ograniczone możliwości i polecenia procesora grafiki**. Ubogie w instrukcje procesory grafiki ograniczają nas co do tego, co możemy zrobić. Dobrze byłoby prowadzić zaburzoną teksturę szumu do funkcji potęgowej, na przykład wykładniczej, ale jesteśmy ograniczeni do prostych operacji arytmetycznych. Poza tym intensywnie wykorzystujemy rendering do tekstury, który nie jest dostępny na wszystkich kartach graficznych. Ponieważ jest pewne, że w przyszłości karty graficzne będą bardziej zaawansowane, będziemy mieli mniej problemów.

Uzyskiwanie większej uniwersalności

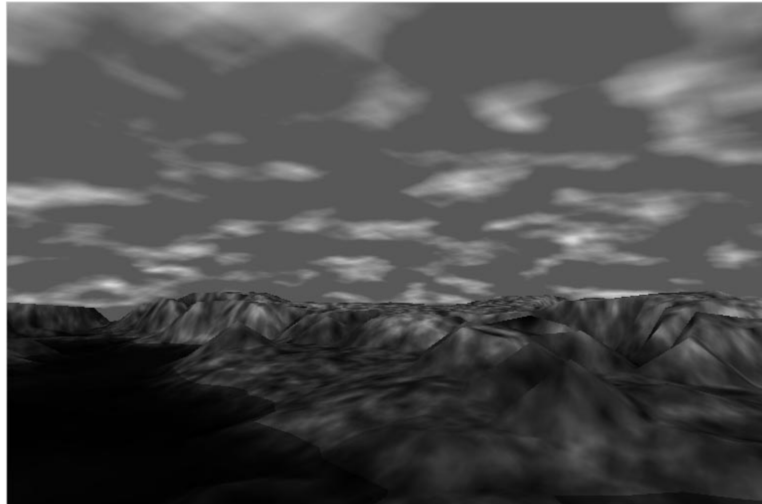
Jeśli chcemy skorzystać z tej techniki w komercyjnym projekcie, a docelową platformą jest komputer klasy PC (gdzie wydajność może się znacznie różnić), musimy zastanowić się nad uniwersalnością tego rozwiązania. Nawet jeśli docelowa platforma jest stała, ale inne elementy gry wymagają większej uwagi, musimy mieć możliwość zmiany wydajności kosztem jakości dla takich przypadków.

Oto kilka sposobów na modyfikację wydajności przedstawionej metody:

- **Rozdzielczość tekstury**. Rozdzielczość tekstury na różnych pośrednich etapach można zmniejszyć, oszczędzając na pamięci i szybkości wypełniania trójkątów. Zauważ, że nie chcemy zmniejszać głębi kolorów, ponieważ przy 16 bitach na kolor jakość drastycznie spada.
- **Częstość aktualizacji tekstury**. Nie każda tekstura musi być aktualizowana w każdej klatce. Szczególnie gdy szybkość symulacji nie jest duża, można zrezygnować z aktualizacji w każdej kratce przedstawionej w przykładowym kodzie.
- **Liczba używanych oktaw**. Przykładowy kod używa czterech oktaw szumu, ale nawet trzy zapewniają dosyć dobre wyniki. Na bardzo szybkich systemach można skorzystać z pięciu oktaw, by podnieść jakość grafiki.

Wnioski

Rysunek 5.4.7 przedstawia wynikowe proceduralne chmury w przykładowym programie.



Rysunek 5.4.7.
Wynik przykładowego programu tworzącego proceduralne chmury

Mamy nadzieję, że ten rozdział przybliżył Ci nieco techniki tworzenia proceduralnych tekstur na przykładzie chmur oraz metody zrzucenia większości obliczeń na procesor karty graficznej.

Sprzęt graficzny staje się coraz szybszy i powoli możemy generować proceduralne tekstury w czasie rzeczywistym. Mamy nadzieję, że wyniki własnych eksperymentów przedstawisz społeczności twórców gier.

Bibliografia

[Ebert98] Ebert D. S. i inni: *Texturing and Modeling: A Procedural Approach*, AP Professional Inc., 1999.

[Elias99] Elias H.: *Cloud Cover*, dostępne w witrynie *http://freespace.virgin.net/hugo.elias/models/m_clouds.htm*, zawiera czysto programową metodę podobną do przedstawionej tutaj.

[Pallister99] Pallister K.: *Rendering to Texture Surface Using DirectX 7*, dostępne w witrynie *www.gamasutra.com/features/19991112/pallister_01.htm*.