

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 3

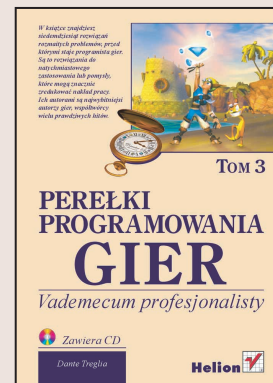
Autor: Dante Treglia

Tłumaczenie: Rafał Jońca

ISBN: 83-7361-111-8

Tytuł oryginału: [Game Programming Gems 3](#)

Format: B5, stron: stron: 728+8 stron kolorowej wkładki



To już trzecia część „Perełek programowania gier. Vademecum profesjonalisty”, wspaniałego zbioru opisów rozmaitych technik programistycznych używanych przez profesjonalnych programistów gier. Twórcy gier i oprogramowania graficznego z firm takich jak Nintendo, ATI, Electronic Arts, Sony Computer Entertainment, Intel, Creative Labs, NVidia, Microsoft, IBM czy Westwood Studios – najlepsi z najlepszych – przedstawiają swoje ulubione algorytmy i sztuczki pozwalające tworzyć gry na światowym poziomie.

Dla początkujących niniejsza książka to prawdziwy przewodnik po wyzwaniach, którym muszą stawić czoła programiści gier. Na końcu każdego rozdziału znajduje się bibliografia, która pozwala rozszerzyć wiadomości o prezentowanym temacie. Jeśli jesteś zaawansowanym programistą, dzięki tej książce zaoszczędzisz mnóstwo czasu. Autorzy spędzili całe miesiące wymyślając, pisząc kod i ilustrując wybrane zagadnienie, więc nie musimy być ekspertami w danym temacie, aby go zrozumieć. Więcej czasu pozostanie na tworzenie wspaniałych funkcji dla własnych gier.

Książka podzielona jest na 6 części:

- Programowanie ogólne
- Matematyka
- Sztuczna inteligencja
- Grafika
- Sieć i gry dla wielu graczy
- Dźwięk

Jeśli marzysz o karierze w przemyśle gier komputerowych, musisz mieć tę książkę. Informacje z pierwszej ręki są zawsze najcenniejsze, a tylko takie informacje znajdziesz w tej książce.



Spis treści

	Słowo wstępne	17
	Przedmowa	21
	Rysunek z okładki	25
	Biografie Współautorów	27
	Część I Programowanie ogólne	43
	Wprowadzenie	45
Rozdział 1.1	Harmonogramowanie zdarzeń gry	47
	Konceptje harmonogramowania	48
	Prosty system harmonogramowania	51
	Zwiększanie złożoności	53
	Podsumowanie	55
	Bibliografia	56
Rozdział 1.2	Szkielet gry wykorzystujący składanie obiektów	57
	Etapy tworzenia gry	57
	Projekt szkieletu gry	59
	Implementacja szkieletu gry	62
	Kod źródłowy	65
	Bibliografia	67
Rozdział 1.3	Zalety makr języka C	69
	Uwaga	69
	Sztuczka 1. — zamiana wyliczeń na tekst	70
	Sztuczka 2. — stałe z reprezentacji binarnych w trakcie kompilacji	71
	Sztuczka 3. — stosowanie opisowych komentarzy w asercjach	72
	Sztuczka 4. — asercje w trakcie kompilacji	72
	Sztuczka 5. — określanie liczby elementów tablicy	73
	Sztuczka 6. — Zamiana <code>__LINE__</code> na tekst	73
	Sztuczka 7. — zabezpieczenie się przed pętlami nieskończonymi	74
	Sztuczka 8. — niewielkie, wyspecjalizowane języki	75
	Sztuczka 9. — upraszczanie interfejsu klasy	76
	Podsumowanie	79
	Bibliografia	79
Rozdział 1.4	Generator kodu dołączania funkcji niezależny od platformy	81
	Młodość i wiedza	81
	Sedno sprawy	82
	Szczegóły	83

	Skrypty.....	85
	Sieć	85
	Podsumowanie	86
	Bibliografia	86
Rozdział 1.5	Inteligentne wskaźniki bazujące na uchwytach	87
	Korzystanie z rozwiązania	88
	Uchwyt.....	88
	Inteligentny wskaźnik	89
	Podsumowanie	91
	Bibliografia	91
Rozdział 1.6	Alokatory STL	93
	Przykład	93
	Podstawy alokatorów	94
	Wymagania stawiane alokatorowi	94
	Domyślny obiekt alokatora	98
	Pisanie własnego alokatora	99
	Sposoby wykorzystania	100
	Dane stanu alokatora	101
	Zalecenia	101
	Szczegóły implementacji	102
	Podsumowanie	102
	Bibliografia	102
Rozdział 1.7	Zapisz mnie teraz!	103
	Dlaczego to jest takie trudne?	103
	Klasa SAVEMGR	104
	Klasa SAVEOBJ	104
	Typy danych i rozszerzenia	105
	Przysyłanie domyślnych funkcji	106
	Prosty przykład	106
	Podsumowanie	107
Rozdział 1.8	Wzorzec projektowy automatycznych list	109
	Implementacja	109
	Komentarz do implementacji	111
	Podsumowanie	113
Rozdział 1.9	Obsługa wyjątków liczb zmiennoprzecinkowych	115
	Dlaczego warto powodować zawieszenia programu?	115
	Czy nasz program obsługuje wyjątki liczb zmiennoprzecinkowych?	116
	Rodzaje wyjątków	116
	Kod	117
	Usuwanie błędów powodujących wyjątki zmiennoprzecinkowe	118
	Podsumowanie	118
Rozdział 1.10	Kreowanie projektu rdzenia gry za pomocą języka UML	119
	Gra to obiekty	119
	Klasy poruszają się... jak skały	122
	Współpraca i iteracja	124
	Zagadnienia implementacyjne	126
	Podsumowanie	127
	Bibliografia	128

Rozdział 1.11	Użycie narzędzi Lex i Yacc do przetwarzania plików danych	129
	Lex	130
	Yacc	130
	Zalety i wady	131
	Współpraca Yacca z Leksem	131
	Pliki danych w podsystemach gry	131
	Integracja Leksa i Yacca z eksporterami danych	133
	Przykład	134
	Podsumowanie	137
	Dostępność narzędzi Flex i Bison	137
	Bibliografia	137
Rozdział 1.12	Tworzenie gier dla rynku światowego	139
	Potencjał rynku	139
	Najważniejsze są podstawy — ekran i wejście	140
	Zbiory znaków	143
	Zagadnienia związane z projektem i interfejsem	145
	Lokalizacja	148
	Planowanie i projektowanie	150
	Testowanie	151
	Podsumowanie	154
	Bibliografia	154
Rozdział 1.13	Wejście i interfejs użytkownika czasu rzeczywistego dla gier 3D	157
	Implementacja interfejsu użytkownika	157
	Określanie elementów interfejsu użytkownika	158
	Zagadnienia związane z lokalizacją	160
	System wejścia	161
	Mysz i joystick	162
	Rola interfejsu użytkownika w walce z opóźnieniami	162
	Podsumowanie	163
	Bibliografia	164
Rozdział 1.14	Dobór naturalny — rozwój menu kołowego	165
	Projektowanie menu kołowego	165
	Ewolucja i badania menu kołowego	166
	Stosowanie menu kołowego	167
	Kierunek rozwoju	172
	Poznajmy miasto SimCity	172
	Życie w domu z The Sims	172
	Podsumowanie	175
	Bibliografia	176
Rozdział 1.15	Krótki kod dziennika zdarzeń bazującego na zasadach	177
	Opisy zasad	177
	Znacznik testowania	178
	Plik konfiguracyjny	179
	Konfigurowalne znaczniki	179
	Dzienniki zdarzeń	179
	Sposób użycia	182
	Podsumowanie	183
	Bibliografia	183

Rozdział 1.16	Usługi dzienników	185
	Zarządzanie informacją	185
	Hierarchia systemu	186
	Interfejs dziennika	189
	Budowanie usługi dziennika	191
	Podsumowanie	194
	Bibliografia	194
Rozdział 1.17	Hierarchiczny program profilujący działający w grze	195
	Drzewo profilowania	196
	Sposób użycia	196
	Implementacja	199
	Podsumowanie	201
	Bibliografia	201
Część II Matematyka		203
	Wprowadzenie	205
Rozdział 2.1	Szybkie funkcje o podstawie 2 dla logarytmów i generatorów liczb pseudolosowych	207
	Logarytm przy podstawie 2 dla liczb całkowitych	207
	Maski bitów i generatory liczb losowych	208
	Sposób działania funkcji	209
	Bibliografia	209
Rozdział 2.2	Stosowanie ułamków wektorowych do tworzenia dokładnej geometrii	211
	Problem	211
	Rozwiązanie — ułamki wektorowe	215
	Korzystanie z ułamków wektorowych	216
	Zakresy wartości	217
	Szczegóły implementacji	218
	Podsumowanie	220
	Bibliografia	220
Rozdział 2.3	Aproksymacja funkcji trygonometrycznych	221
	Pomiar błędu	221
	Sinus i cosinus	222
	Aproksymacja wielomianami	229
	Uwagi na temat zbieżności	234
	Podsumowanie	235
	Bibliografia	235
Rozdział 2.4	Kompresja kwaternionów	237
	Kwaterniony	237
	Metoda trzech najmniejszych	237
	Metoda biegunowa	238
	Implementacja	239
	Wydajność	240
	Podsumowanie	241
	Podziękowania	241
	Bibliografia	241

Rozdział 2.5	Ograniczona kinematyka odwrotna (IK)	243
	Hierarchia kości	243
	Cykliczne dochodzenie do współrzędnych	244
	Ograniczanie obrotu	246
	Wykonanie obrotu i zastosowanie ograniczeń	246
	Podsumowanie	249
	Bibliografia	249
Rozdział 2.6	Modelowanie świata za pomocą automatu komórkowego	251
	Podstawy automatów komórkowych	252
	Drzewo ósemkowe	255
	Fizyka praktyczna	255
	Model rdzenia przetwarzania	256
	Powietrze	257
	Woda	257
	Przepływ	258
	Ciepło	259
	Ogień	261
	Dynamiczna częstotliwość aktualizacji	263
	Podsumowanie	264
	Bibliografia	264
Rozdział 2.7	Ruch z tarcie w symulacjach dynamicznych	267
	Tarcie	267
	Metody numeryczne	271
	Wzory dla przestrzeni trójwymiarowej	276
	Zagadnienia geometryczne	277
	Podsumowanie	278
	Bibliografia	278
Część III Sztuczna inteligencja		279
	Wprowadzenie	281
Rozdział 3.1	Optymalizacja uczenia za pomocą techniki GoCap	283
	Omówienie architektury systemu GoCap	283
	Nauka prowadzenia samochodu	285
	Uczenie zasad	289
	Podsumowanie	291
	Bibliografia	291
Rozdział 3.2	Nawigacja obszarowa — rozszerzanie zagadnienia znajdowania drogi	293
	Zacząć od starego podejścia	294
	By uzyskać nowe	295
	Dziel i rządź	298
	Przechodzenie przez ścieżkę	302
	Rozszerzenie zagadnienia	306
	Podsumowanie	307
	Bibliografia	308
Rozdział 3.3	Automat stanów skończonych wykorzystujący wskaźniki na funkcje	309
	Czym jest automat stanów?	309
	Implementacja automatu stanów	311

	Implementacja — klasa CFSM.....	313
	Korzystanie z klasy CFSM.....	316
	Podsumowanie.....	319
	Bibliografia.....	319
Rozdział 3.4	Analiza terenu w grach RTS — ukryta siła	321
	Obszary.....	321
	Wielokąty wypukłe.....	326
	Siła pudełka od zapalek.....	329
	Punkty przyciągania.....	332
	Przeprowadzanie analizy terenu.....	336
	Podsumowanie.....	336
	Bibliografia.....	336
Rozdział 3.5	Rozszerzalny system zdarzeń dla agentów sztucznej inteligencji, obiektów i zadań.....	337
	Wprowadzenie do systemu wyzwalań.....	337
	System wyzwalań dla obiektów.....	338
	Definiowanie warunków.....	338
	Łączenie warunków logiką boolowską.....	339
	Definiowanie odpowiedzi.....	340
	Obliczanie wyzwolenia.....	341
	Pojedyncze strzały i czas przeładowania.....	342
	Łączenie wyzwalań z licznikami i znacznikami.....	342
	Systemy wyzwalań a języki skryptowe.....	343
	Ograniczenia.....	344
	Podsumowanie.....	345
	Bibliografia.....	345
Rozdział 3.6	Znajdowanie najbardziej odpowiedniej taktycznie drogi algorytmem A*	347
	A*, ale bardziej ryzykownie.....	348
	Poprawianie niedoskonałej ścieżki.....	350
	Czas wystawienia i modelowanie wroga.....	351
	Zagrożenia nie są statyczne.....	353
	Poprawa uzyskiwanych ścieżek.....	353
	Wydajność.....	354
	Wydajne testy widzialności i ostrzału.....	354
	Rozszerzone koszty A*.....	356
	Program ASE.....	357
	Podsumowanie.....	357
	Bibliografia.....	357
Rozdział 3.7	Szybka metoda obliczeń dla siatek nawigacyjnych	359
	Przeszkody dynamiczne i statyczne.....	359
	Siatki nawigacyjne.....	360
	Portale.....	360
	Budowanie tablicy.....	363
	Pozostałe zagadnienia dotyczące portali.....	365
	Reprezentacja postaci.....	365
	Przeszkody dynamiczne.....	366
	Poruszanie się między przeszkodami statycznymi i dynamicznymi.....	369
	Dodatkowe uwagi dotyczące siatek nawigacyjnych.....	369
	Podsumowanie.....	370

Rozdział 3.8	Wybór związku między znajdowaniem drogi a zderzeniami	371
	Ruch pod kontrolą zderzeń	371
	Model zderzeniowy dla znajdowania drogi	372
	Podejście 1. Sztuczna inteligencja z tolerancją błędów	372
	Podejście 2. Znajdowanie drogi w podzbiórze niezajętej przestrzeni	375
	Podejście 3. Korzystanie ze znajdowania drogi w zderzeniach postaci	377
	Implementacja ruchu wzdłuż ścieżki	378
	Podsumowanie	380
	Bibliografia	381
Część IV Grafika	383	
	Wprowadzenie	385
Rozdział 4.1	Eliminacja połączeń typu T i przeliczanie siatek.....	389
	Eliminacja połączeń typu T	390
	Ponowna triangulacja.....	391
	Implementacja.....	392
	Podsumowanie	393
Rozdział 4.2	Szybkie obliczenia normalnych dla pól wysokości.....	395
	Normalne dla dowolnych siatek	395
	Normalne pól wysokości	396
	Podsumowanie	399
	Kod źródłowy	399
	Bibliografia	399
Rozdział 4.3	Szybkie normalne dla łąt	401
	Definicje	401
	Podejścia tradycyjne	402
	Inne rozwiązania	402
	Prostszy sposób.....	402
	Inne zalety.....	403
	Jaką dokładność uzyskujemy?.....	403
	Podsumowanie	403
	Bibliografia	404
Rozdział 4.4	Szybkie i proste wykrywanie zasłaniania.....	405
	Problem widzialności.....	405
	Algorytm PLP	406
	Algorytm cPLP	407
	Zalety algorytmów	408
	Wyniki eksperymentów	408
	Podsumowanie	409
	Bibliografia	410
Rozdział 4.5	Tworzenie, optymalizacja i rendering pasków trójkątów.....	411
	Paski trójkątów	411
	Tworzenie pasków trójkątów	413
	Optymalizacja	416
	Rendering	417
	Paski trójkątów przyjazne dla pamięci podręcznej.....	417
	Paski dla ciągłych poziomów szczegółowości	417
	Podsumowanie	418
	Bibliografia	418

Rozdział 4.6	Obliczanie zoptymalizowanych obszarów cienia dla złożonych zestawów danych	419
	Poprzednie algorytmy	419
	Algorytm	420
	Algorytm optymalizacji	421
	Bibliografia	423
Rozdział 4.7	Podział powierzchni w animacji postaci.....	425
	Sposoby podziału	425
	Hierarchia kości i bufor akumulacji wierzchołków	430
	Optymalizacje	431
	Łączymy wszystko	433
	Kod źródłowy	435
	Podsumowanie	435
	Bibliografia	435
Rozdział 4.8	Udoskonalona deformacja kości	437
	Tło	437
	Proste metody	438
	Dodawanie kości	438
	Zmiana wag	439
	Łączymy wszystko i przyspieszamy działanie	443
	Podsumowanie	445
	Bibliografia	445
Rozdział 4.9	Rdzeń systemu realistycznego poruszania postacią.....	447
	Problem — przejście do dowolnego celu	448
	Problem — płynne przejście między animacjami	449
	Rdzeń nowego rozwiązania — lokalne modyfikatory z niezależnymi stosunkami przejść	451
	Zastosowanie — ruch do dowolnego celu	452
	Modyfikatory przemieszczenia	453
	Zastosowanie — przejścia	454
	Dodatkowe informacje	455
	Podsumowanie	455
	Bibliografia	455
Rozdział 4.10	Kompilator programowalnego shadera wierzchołków	457
	Programowalny shader wierzchołków	457
	Kompilator	459
	Składniki kompilatora	460
	Podsumowanie	464
	Podziękowania	464
	Bibliografia	464
Rozdział 4.11	Promienie z billboardów	467
	Macierze	467
	Wierzchołki	468
	Mapowanie UV	468
	Podsumowanie	470
Rozdział 4.12	Sztuczki trójwymiarowe w izometrycznych rdzeniach gier	471
	Przejście do trzeciego wymiaru	472
	Podsumowanie	477
	Bibliografia	477

Rozdział 4.13	Symulacja krzywizn za pomocą map normalnych.....	479
	Mapy normalnych.....	479
	Opis procesu.....	480
	Przygotowanie danych.....	481
	Rzucanie promieni.....	481
	Uzyskiwanie szczegółów.....	483
	Przetwarzanie końcowe.....	483
	Często spotykane problemy.....	484
	Inne podejście.....	485
	Podsumowanie.....	485
	Podziękowania.....	485
	Bibliografia.....	486
Rozdział 4.14	Metody dynamicznego, realistycznego oświetlenia terenu	487
	Tło.....	487
	Taksonomia rozwiązań.....	489
	Światło słoneczne — kąt horyzontu, elipsy cienia i wielomianowe mapy tekstur.....	489
	Światło z nieba — łąty i aproksymacja metodą radiosity.....	492
	Animowane cienie chmur.....	493
	Rozwiązanie bazujące na sekwencjach wideo.....	495
	Obiekty, które nie są terenem.....	496
	Podsumowanie.....	496
	Bibliografia.....	496
Rozdział 4.15	Metoda oświetlenia używająca map sześciennych	499
	Fizyczne właściwości map sześciennych.....	499
	Jak pobrać dane z mapy sześciennnej lub umieścić je na niej.....	500
	Rendering map sześciennych.....	501
	Kodowanie pokrywy chmur.....	502
	Kodowanie światła na mapie sześciennnej.....	504
	Kodowanie światła rozmytych na mapach sześciennych.....	504
	Kodowanie cyklu dnia i nocy na mapie sześciennnej.....	505
	Podsumowanie.....	505
	Bibliografia.....	506
Rozdział 4.16	Teksturowanie proceduralne	507
	Parametry i procedury.....	507
	Skupiamy się na grach.....	508
	Akceleracja sprzętowa.....	511
	Podsumowanie.....	511
	Podziękowania.....	512
	Bibliografia.....	513
Rozdział 4.17	Unikalne tekstury	515
	Tekstury proceduralne.....	515
	Inteligentne buforowanie tekstur.....	516
	Model łączy.....	516
	Mapowanie warstw i przekształcenia.....	516
	Źródła i filtry warstw.....	517
	Metody łączenia.....	518
	Wartości sterujące.....	518
	Tekstury dynamiczne.....	519
	Skalowalność.....	519
	Łączenie wykonywane w procesorze czy karcie graficznej?.....	520
	Demo.....	521

	Podsumowanie.....	522
	Bibliografia.....	522
Rozdział 4.18	Tekstury jako tablice przeglądowe w obliczeniach oświetlenia dla pikseli.....	523
	Oświetlenie z rozbłyskiem dla pikseli bez korzystania z mapy sześciennej w celu normalizacji (mapowanie n.h/h.h).....	523
	Wykładnik rozbłysku dla pikseli na podstawie mapy (n.h) ^k	525
	Opalizacja przesunięcia kolorów.....	528
	Światła punktowe z poprawnym zanikiem oparte na pikselach.....	529
	Światła kierunkowe i reflektory z poprawnym zanikiem oparte na pikselach.....	530
	Podsumowanie.....	531
	Bibliografia.....	531
Rozdział 4.19	Rendering wykorzystujący ręcznie tworzone modele cieniowania.....	533
	Modele cieniowania.....	533
	Model cieniowania bazujący na mikrofasetach.....	534
	Cieniowanie NDF.....	535
	Mapowanie nierówności w NDF.....	537
	Rozszerzenia.....	538
	Podsumowanie.....	538
	Bibliografia.....	538
	Część V Sieć i gry dla wielu graczy.....	541
	Wprowadzenie.....	543
Rozdział 5.1	Minimalizacja opóźnień w grach RTS.....	545
	Blokowanie klatek a blokowanie zdarzeń.....	545
	Synchronizacja czasu.....	549
	Podsumowanie.....	551
	Bibliografia.....	552
Rozdział 5.2	Protokół sieciowy dla strategii czasu rzeczywistego.....	553
	Inne protokoły.....	553
	Nasz protokół.....	555
	Udoskonalanie.....	557
	Użyteczne moduły.....	559
	Pułapki w grze StarTopia.....	560
	Przykładowa gra.....	562
	Podsumowanie.....	562
	Bibliografia.....	562
Rozdział 5.3	Elastyczna architektura symulacji dla gier z ogromną liczbą graczy.....	563
	Opis architektury.....	563
	Klasy pomocnicze.....	564
	Klasy podstawowe.....	567
	Zarządcy i fabryki.....	570
	Łączymy wszystko.....	572
	Podsumowanie.....	574
	Bibliografia.....	574

Rozdział 5.4	Skalowalne serwery gier sieciowych	577
	Strategie zapewniające grę w pełni fair	577
	Projektowanie skalowalnych serwerów	579
	Rozkład obciążenia	584
	Optymalizacja	587
	Podsumowanie	590
	Bibliografia	590
Rozdział 5.5	Serializacja obiektów bazująca na szablonach	591
	Istniejące rozwiązania	591
	Przenośność	593
	Klasa Serializer	594
	Możliwe rozszerzenia i optymalizacje	599
	Dalsze rozszerzanie możliwości	601
	Podsumowanie	602
	Bibliografia	602
Rozdział 5.6	Bezpieczne gniazda	603
	Standard IPSec	603
	Założenia	604
	Związki bezpieczeństwa	604
	Format pakietu	605
	Wysyłanie danych	607
	Otrzymywanie danych	608
	Przykładowa implementacja	610
	Interfejs CryptoAPI	611
	Wydajność	612
	Bezpieczeństwo	613
	Podsumowanie	613
	Bibliografia	613
Rozdział 5.7	Narzędzie symulacji i monitorowania sieci	615
	Interfejs	615
	Monitoring sieci	616
	Symulacja połączeń TCP	617
	Symulacja połączeń UDP	617
	Symulacja przepustowości komputera	618
	Podsumowanie	618
Rozdział 5.8	Tworzenie gier dla wielu graczy z wykorzystaniem DirectPlay 8.1	619
	Architektura systemu DirectPlay	619
	Transmisja danych	620
	Wywołania zwrotne	624
	Wysyłanie głosu za pomocą DirectPlay	627
	Źródła informacji	629
Rozdział 5.9	Gry w sieciach bezprzewodowych na podstawie środowiska Java Micro Edition	631
	Charakterystyka sieci	632
	Środowisko Java Micro Edition	633
	Obsługa sieci w J2ME	633
	Ograniczenia protokołu HTTP	635
	Optymalizacja pakietów	636

	Pobieranie obrazów z serwera	637
	Podsumowanie	638
	Bibliografia	639
Część VI Dźwięk.....		641
	Wprowadzenie	643
Rozdział 6.1 Kompresja dźwięku przy użyciu Ogg Vorbis		645
	Kompresja psychoakustyczna.....	645
	Sposoby stosowania kompresji.....	647
	Przykłady kodu korzystającego z Ogg Vorbis.....	648
	Podsumowanie.....	651
	Bibliografia	652
Rozdział 6.2 Tworzenie realistycznego środowiska trójwymiarowego.....		653
	Podstawowe koncepcje systemu dźwięku trójwymiarowego.....	653
	Wydajne wykorzystanie systemu dźwięku	654
	Implementacja.....	656
	Podsumowanie	657
	Materiały.....	657
Rozdział 6.3 Przeszkody jako prostopadłościany wyrównane do osi		659
	Problem	659
	Rozwiązanie.....	660
	Implementacja.....	663
	Podsumowanie	664
	Bibliografia	664
Rozdział 6.4 Korzystanie z dwukwadratowego filtra rezonansowego.....		665
	Zasada działania filtrów cyfrowych.....	665
	Filtry IIR a filtry FIR	666
	Implementacja filtra dwukwadratowego	666
	Permutacja zmiennych.....	667
	Unikanie denormalizacji	668
	Sterowanie filtrem	668
	Obliczanie współczynników filtra	668
	Filtr dolnoprzepustowy	669
	Filtr górnoprzepustowy.....	669
	Filtr pasmowoprzepustowy.....	670
	Szeregowe łączenie filtrów	670
	Równoległe łączenie filtrów	670
	Oprogramowanie	670
	Podsumowanie	671
	Bibliografia	671
Rozdział 6.5 Kodowanie z przewidywaniem liniowym dla kompresji dźwięku i efektów		673
	Modelowanie głosu.....	674
	Symulacja programowa	675
	Zastępowanie strun głosowych.....	677
	Sterowanie ponowną syntezą.....	678
	Zwiększenie głębi rozmówcy	679
	Kodowanie danych	680
	Szybkość	680

	Eksperymenty	681
	Bibliografia	681
Rozdział 6.6	Synteza stochastyczna dla złożonych dźwięków	683
	Algorytm przystawiania liniowego	683
	Rodzaje szumu	684
	Przykłady	685
	Generowanie odgłosów deszczu	686
	Podsumowanie	689
	Bibliografia	690
Rozdział 6.7	Przetwarzanie dźwięku w grach w czasie rzeczywistym.....	691
	Modułowe przetwarzanie dźwięku	691
	Proceduralna generacja dźwięku	692
	System Sphinx MMOS	692
	Procesory	693
	Opis pliku drogi	693
	Zastosowania plików dróg dźwięku	695
	Kod źródłowy	698
	Podsumowanie	699
	Bibliografia	699
Dodatki.....		701
	Zawartość płyty CD	703
	Skorowidz	705

Rozdział 4.5

Tworzenie, optymalizacja i rendering pasków trójkątów

Carl S. Marshall, Intel Labs

Carl.S.Marshall@intel.com

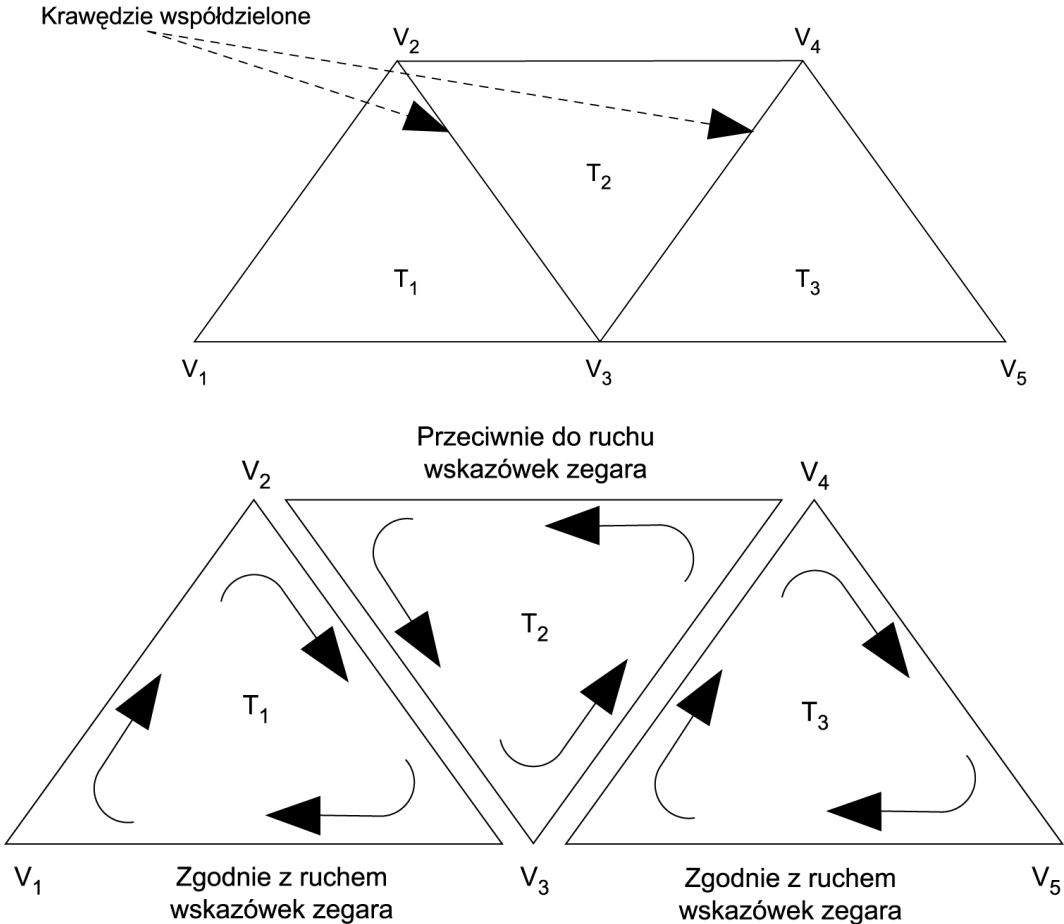
Obecnie, gdy istnieją wysokiej wydajności konsole do gier, paski trójkątów są stosowane coraz częściej do reprezentacji i renderingu geometrii. W tym rozdziale zajmę się zagadnieniem generowania pasków trójkątów z dowolnych modeli trójwymiarowych. Po omówieniu algorytmu pasków trójkątów przedstawię zalety ich stosowania, a także błędy, które mogą pojawić się przy ich kreowaniu. Następnie zajmę się sposobami ich przekazywania do interfejsu programistycznego kart graficznych. Poza tym opiszę i poddam konstruktywnej krytyce kilka innych algorytmów dotyczących trójkątów.

Paski trójkątów

Pasek trójkątów to ciąg trójkątów połączonych ze sobą. Połączenie trójkątów umożliwia buforowanie wierzchołków w karcie graficznej, a także ponowne wykorzystywanie współdzielonych krawędzi trójkąta. Rysunek 4.5.1 przedstawia prosty pasek trójkątów ze współdzielonymi krawędziami V_2V_3 i V_3V_4 . Aby trójkąt mógł należeć do paska, musi mieć tę samą grupę wygładzania i materiału, co pozostałe trójkąty. Grupa wygładzania to grupa trójkątów posiadających tę samą normalną dla wierzchołków, a grupa materiału to grupa trójkątów o takim samym oświetleniu i właściwościach tekstury.

Historia

Paski trójkątów są stosowane od dłuższego czasu. Wcześniej do funkcji karty graficznej jawnie wysyłało się położenie, normalną i kolor każdego wierzchołka. Ponieważ jednak dla pasków trójkątów nie musimy przysyłać wierzchołków, które się powtarzają, zmniejszamy zapotrzebowanie na przepustowość magistrali. Teraz indeksowanie wierzchołków zastąpiło wcześniejszy proces i stało się podstawową metodą przekazywania wielokątów do karty graficznej. [Marselas00] sugeruje, że korzystanie z pasków trójkątów i indeksowania wierzchołków umożliwia uzyskanie stosunku wierzchołek-trójkąt wynoszącego prawie 1:1. W ten sposób znacznie redukuje się ilość danych.



Rysunek 4.5.1. (A) Prosty pasek trójkątów z współdzielonymi krawędziami V_2V_3 i V_3V_4 . (B) Kolejność trójkątów w pasku. Wierzchołki numerowane są naprzemiennie zgodnie z ruchem wskazówek zegara i przeciwnie do niego

Cele

Paski trójkątów będziemy generować z myślą o czterech celach:

1. Aby zminimalizować liczbę pasków trójkątów.
2. Aby zminimalizować liczbę powtarzających się wierzchołków.
3. Aby zminimalizować liczbę trójkątów swobodnych.
4. Aby zmaksymalizować wydajność buforowania wierzchołków.

Przedstawione cele zazwyczaj są ze sobą sprzeczne. Na przykład bardzo trudno jest wygenerować bardzo długi pasek trójkątów przyjazny dla buforu bez powtarzania znacznej liczby wierzchołków. Czasem warto pozostawić niektóre trójkąty samym sobie, zamiast starać się za wszelką cenę dołączyć je do paska, bo będzie to rozwiązanie bardziej wydajne.

Korzyści

Korzystanie z pasków trójkątów zamiast niezależnych trójkątów umożliwia redukcję zbędnych wierzchołków i ich indeksów. W zależności od sposobu traktowania pasków trójkątów przez sprzęt, możemy zaoszczędzić na danych wierzchołków, oświetleniu i przekształceniach. Poza tym paski zdecydowanie bardziej efektywnie wykorzystują buforowanie wierzchołków na karcie graficznej.

Tworzenie pasków trójkątów

Istnieje kilka opracowanych przez naukowców algorytmów tworzenia pasków trójkątów i oczywiście każdy z nich ma swoje zalety i wady, jeśli chodzi o znajdowanie optymalnych pasków. [Evans96a] używa siatki bazującej na czworokątach, aby zoptymalizować paski wewnątrz łąt, natomiast [Hoppe99] stosuje podejście przyjazne dla buforów wierzchołków karty graficznej. Rozwiązanie, którym się zajmę, optymalizuje długość paska trójkątów, aby zlikwidować narzut powodowany przez interfejs programistyczny grafiki.

Ponieważ niemożliwe jest osiągnięcie idealnego paska trójkątów dla każdej siatki, należy wymyślić algorytm znajdowania najbardziej optymalnych pasków dla danego zestawu danych. Zastosowany algorytm generuje paski o jak największej długości dla dowolnego modelu trójwymiarowego złożonego z wielokątów. Generowanie pasków może się odbywać w aplikacji do kreowania modeli trójwymiarowych lub stanowić osobne narzędzie. Naszym celem jest uzyskanie długich pasków, a następnie zapisanie ich w formacie ułatwiającym ich rendering. Kolorowa wkładka 5. przedstawia przykładowy obraz dwóch modeli po przeprowadzeniu podziału na paski trójkątów.

Definicje

Zanim przejdziemy do algorytmu tworzenia pasków trójkątów, musimy zdefiniować kilka terminów. *Aktywna krawędź* to krawędź trójkąta wewnątrz paska, do której dodawane są nowe trójkąty. Znajduje się ona między drugim a trzecim wierzchołkiem ostatniego dodanego paska trójkątów. Pogrubiona krawędź z rysunku 4.5.2a przedstawia aktywną krawędź, DE, do której zostanie dodany trójkąt z wierzchołkiem F. *Zamiana* używana jest wtedy, gdy aktywna krawędź nie jest wyrównana z sąsiednim trójkątem dodawanym do paska. Rysunek 4.5.2b przedstawia sytuację, w której aktywną krawędzią jest DE, ale następny trójkąt jest dodawany do krawędzi CE. Aby dodać nowy trójkąt, wierzchołek C musi zostać powtórzony, a następnie zamieniony z wierzchołkiem E. W ten sposób zachowamy przemiennosc kierunku — zgodnie i przeciwnie do ruchu wskazówek zegara. Ostatni termin, który zdefiniujemy, to *obrócenie*. Stosujemy go, gdy duplikujemy i zamieniamy dwa wierzchołki, aby dodać nowy (patrz rysunek 4.5.2c).

Przygotowywanie danych

Etap przygotowań dotyczy utworzenia metryki pozwalającej wykonać paski trójkątów odpowiedniej jakości.

Rysunek 4.5.2.

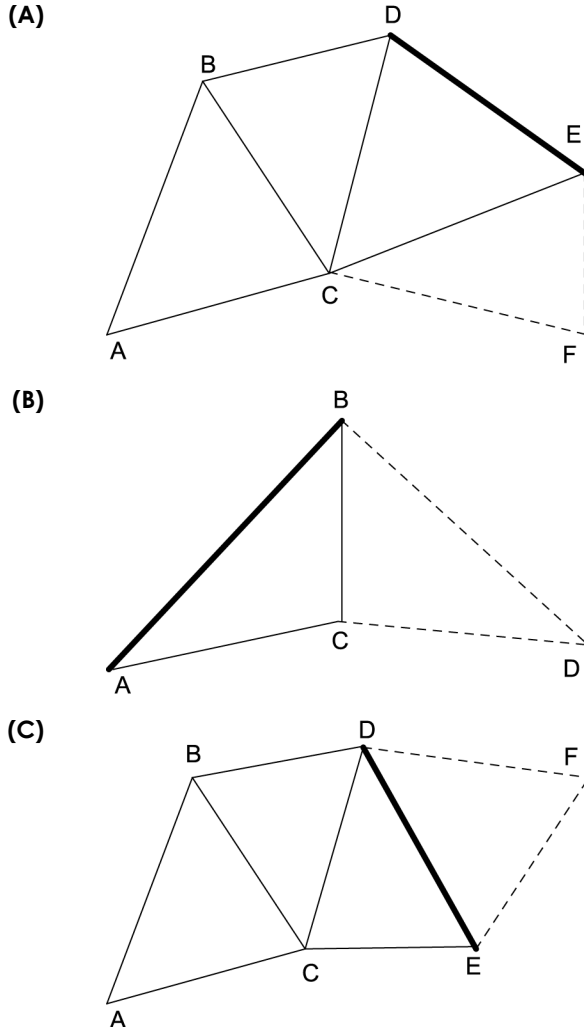
Różne sposoby dodawania trójkątów do paska.

(A) Pasek z indeksem wierzchołków w kolejności $ABCDE$ i aktywną krawędzią DE . Ponieważ aktywna krawędź dotyka nowego trójkąta, dodajemy po prostu wierzchołek F na końcu paska. Nowy pasek składa się teraz z wierzchołków $ABCDEF$.

(B) Pasek z indeksem wierzchołków w kolejności $ABCDE$ i aktywną krawędzią DE . Aby dodać wierzchołek F , musimy zduplikować C i zamienić z E , ponieważ nowy trójkąt nie styka się z aktywną krawędzią. Nowy pasek składa się teraz z wierzchołków $ABCDCEF$.

(C) Przypadek, w którym zmieniamy kolejność pierwszej powierzchni, by dopasować ją do drugiego trójkąta jest istotny, jeśli chcemy zminimalizować powtarzanie się wierzchołków. Jeśli pasek trójkątów zaczął się jako CAB , musielibyśmy powtórzyć B i C , aby dodać wierzchołek D . Nowy pasek składa się z wierzchołków $CABBCD$.

Jeśli aktywna krawędź nie styka się z nowym trójkątem, musimy powtórzyć dwa wierzchołki



1. Znajdujemy trójkąt o najmniejszym polu. Nazwiemy go trójkątem początku. Aby uniknąć słabego rozplanowania pasków, powinniśmy znaleźć 10 najmniejszych trójkątów siatki i spośród nich wybrać najbardziej dogodny do rozpoczęcia obliczeń.
2. Wybieramy wierzchołek trójkąta początku lub obliczamy jego środek, który stanie się pierwszym wierzchołkiem dla algorytmu pasków trójkątów.
3. Tworzymy środki wszystkich trójkątów. Możemy to łatwo wykonać, uśredniając trzy wierzchołki każdego trójkąta (patrz równanie 4.5.1). C jest położeniem środka, a V_1 , V_2 i V_3 to położenia wierzchołków trójkąta.
4. Dla każdego trójkąta obliczamy i przechowujemy odległość między wierzchołkiem początku i środkiem.

$$C = (V_1 + V_2 + V_3) / 3, 0 \quad (4.5.1)$$

Algorytm tworzenia

Po realizacji etapu przygotowań możemy rozpocząć generowanie pasków trójkątów. Po dodaniu trójkąta siatki do paska oznaczamy taki trójkąt jako wykorzystany.

1. Zaznaczamy poprawny trójkąt. Będzie to pierwszy trójkąt paska.
2. Jeśli trójkąt ma sąsiadów, wybieramy spośród nich ten o najmniejszej odległości i czynimy go aktualnym trójkątem. W przeciwnym razie kończymy tworzenie paska i przechodzimy do kroku 6.
3. Zmieniamy kolejność wierzchołków pierwszego trójkąta w taki sposób, aby dopasować aktywną krawędź do drugiego trójkąta. Drugiego trójkąta powinna dotykać krawędź uzyskiwana między drugim i trzecim wierzchołkiem.
4. Pobieramy sąsiada aktualnego trójkąta, znajdując najbliższego sąsiada, którego odległość została policzona w kroku 4. etapu przygotowań. Jeśli trójkąt nie jest wyrównany do aktywnej krawędzi, musimy dokonać wstawienia wierzchołka i zamiany, aby kontynuować pasek (patrz rysunek 4.5.2c).
5. Przechodzimy do kroku 2.
6. Sprawdzamy, czy pozostały jeszcze jakieś trójkąty niezwiązane z żadnym paskiem. Jeśli tak, przechodzimy do kroku 1.

Pseudokod wysokiego poziomu dla algorytmu pasków przedstawia listing 4.5.1.

Listing 4.5.1. Pseudokod wysokiego poziomu dla algorytmu tworzenia pasków trójkątów

```
void main( )
{
    Mesh *pMesh;

    LoadMesh(pMesh); // wczytanie trójwymiarowej siatki wielokątów
    originTriangle = FindSmallestAreaTriangle();
    CalculateCentroidForEachTriangle();

    // obliczenie odległości od początkowego trójkąta
    // do aktualnego trójkąta
    CalculateDistanceFromToEachTriangle(OriginTriangle);

    // tworzenie pasków trójkątów
    TriStripGeneration(pMesh);

    // wykonanie filtrowania w celu sprawdzenia, czy wcześniej
    // wygenerowane paski można połączyć
    ConnectTriangleStrips();
    ConvertTriStrips(); // Zastosowanie własnej struktury danych
}
```

Po zakończeniu algorytmu tworzenia uzyskujemy N pasków trójkątów. Każdy z pasków należy do tej samej grupy wygładzania i ma ten sam identyfikator materiału. Po umieszczeniu pasków w pamięci możemy je zapisać w formacie wymaganym przez grę.

Łączenie pasków trójkątów

W drugim przebiegu analizujemy paski, by sprawdzić, czy któreś z punktów tworzących jeden pasek mogą zostać połączone z innym paskiem. Analizę zaczynamy od określenia wszystkich krawędzi początkowych i końcowych każdego paska trójkątów, a także indeksów powierzchni związanych z każdą z nich. Następnie dokonujemy porównania tych krawędzi z takimi samymi krawędziami innych pasków. Jeśli znajdziemy dopasowanie, wtedy dwa paski trójkątów są analizowane, by dokonać połączenia (patrz rysunek 4.5.2). W większości sytuacji w trakcie połączenia będziemy musieli powtórzyć niektóre wierzchołki. Najprostszy przypadek ma miejsce wtedy, gdy znajdziemy dopasowanie paska do pojedynczego trójkąta. W większości sytuacji możemy tak zmienić kolejność wierzchołków, by dodać ten trójkąt do paska.

Optymalizacja

Ponieważ paski trójkątów tworzymy po to, by zwiększyć wydajność, optymalizacja to kluczowy element algorytmu generowania pasków. Optymalizację można przeprowadzić na kilku etapach procesu tworzenia pasków trójkątów — podczas początkowego przetwarzania, generowania i wykorzystywania w czasie rzeczywistym.

Początkowe przetwarzanie

- Tworzymy siatki z tylko kilkoma grupami wygładzania (więcej niż jedna normalna na wierzchołek).
- Ograniczamy liczbę grup materiałów dla modelu.
- Optymalizujemy model w taki sposób, by powierzchnie wielokątów nie miały naprzemiennie ułożonych grup materiałów lub wygładzania, ponieważ ograniczyłyby to długość pasków trójkątów.
- Sortujemy paski względem grup materiałów przed ich przekazaniem do karty graficznej.
- Eliminujemy wszystkie niepotrzebne powierzchnie. Taka powierzchnia istnieje, gdy dwa (lub więcej) z trzech położań wierzchołków są równoważne. Może to powodować błędy w renderingu, gdy będą stosowane paski.

Generowanie

- Osobne trójkąty umieszczamy w jednym poindeksowanym buforze.
- Staramy się połączyć paski, których trójkąty początku i końca znajdują się obok siebie.
- Powinniśmy znać rozmiar pamięci podręcznej sprzętu, by odpowiednio ją wykorzystać.

W trakcie działania

- Używamy wywołań interfejsu dotyczących poindeksowanej tablicy zamiast wywołań dotyczących kolejnych wierzchołków.
- Redukujemy zmiany stanów i dynamicznych współrzędnych tekstur.

Rendering

Większość interfejsów programistycznych kart graficznych obsługuje paski trójkątów. Istnieje kilka formatów, w jakich możemy przesyłać paski do karty graficznej. Jeden z nich wymaga wysłania danych wierzchołka dla każdego wierzchołka paska. Jest to bardzo kosztowne, ponieważ wiele wierzchołków jest zduplikowanych z racji współdzielonych krawędzi wewnątrz paska. Inny format wykorzystuje przesyłanie pasków w postaci indeksów. Oznacza to, że najpierw przekazujemy dane wierzchołków dla danego fragmentu wraz z indeksami, a następnie korzystamy już tylko z indeksów. Problem polega na tym, że w większości kart graficznych w jednym wywołaniu funkcji możemy określić tylko jeden trójkąt. Niektóre systemy umożliwiają jednak przekazanie całych pasków w jednym wywołaniu, jeśli stosuje się dodatkowe wierzchołki. W [Neider99] i [Microsoft00] znajdują się listy funkcji, z których należy korzystać w przypadku pasków trójkątów. Biblioteka OpenGL wykorzystuje typ `GL_TRIANGLE_STRIP`, a biblioteka Direct3D firmy Microsoft — typ `D3DPT_TRIANGLESTRIP`.

Paski trójkątów przyjazne dla pamięci podręcznej

Inne algorytmy tworzenia pasków trójkątów starają się być przyjazne dla pamięci podręcznej, ograniczając chybienia w buforze wierzchołków karty graficznej [Hoppe99]. [Nvidia00] stosuje specjalny schemat rozkładu wierzchołków, który wykonuje się po utworzeniu pasków trójkątów. Zaletą tego podejścia jest to, że możemy zoptymalizować trójkąty w sposób korzystny dla karty graficznej. Oczywiście rozwiązanie to ma także wadę — optymalizacja będzie dotyczyła danego rodzaju kart graficznych i może dać efekt odwrotny od zamierzonego w przypadku innych kart.

Paski dla ciągłych poziomów szczegółowości

Tworzenie pasków trójkątów dla siatek o ciągłym poziomie szczegółowości nie jest proste. Usunięcie jednego wierzchołka lub powierzchni z siatki może mieć znaczny wpływ na paski, jeśli były tworzone dla siatki o wysokiej rozdzielczości. Istnieje kilka sposobów rozwiązania tego problemu. Pierwszy polega na wygenerowaniu pasków dla każdej rozdzielczości modelu. Jest to bardzo niepraktyczne i może znacznie zwiększyć zapotrzebowanie na pamięć. Drugie rozwiązanie to tworzenie pasków „w locie” i przechowywanie ich w pamięci aż do zmiany rozdzielczości siatki. Wybieramy drugi sposób i stawiamy jeden warunek: trójkąty mogą być dodawane tylko do sąsiadów aktywnej krawędzi lub możliwa jest zamiana. Jeśli nie istnieją sąsiedzi, kończymy pasek. Musimy tak zoptymalizować strukturę danych, by móc szybko wyszukiwać i sprawdzać sąsiadów.

Zaletą tego rozwiązania jest możliwość stosowania pasków dla siatek generowanych dynamicznie — niestety zwiększy to zapotrzebowanie na pamięć.

Podsumowanie

W tym rozdziale przedstawiłem, jak tworzyć i optymalizować paski trójkątów dla gier. Gdy zamierzamy renderować siatki, paski zapewniają zwiększenie szybkości w porównaniu z przekazywaniem zwykłych list trójkątów. Zachęcam do sprawdzenia wersji oryginalnej i z paskami, aby samemu przekonać się o występujących między nimi różnicach.

Bibliografia

[Evans96a] Evans Francine, Steven Skiena i Amitabh Varshney: „Optimizing Triangle Strips for Fast Rendering”, *Visualization '96 Proceedings*, IEEE, 1996, strony 319 – 326.

[Evans96b] Evans Francine, Steven Skiena i Amitabh Varshney: „Completing Sequential Triangulations Is Hard”, sprawozdanie techniczne, *Department of Computer Science*, State University of New York at Stony Brook, 1996.

[Hoppe99] Hoppe Hughes: „Optimization of Mesh Locality for Transparent Vertex Caching”, *Computer Graphics Proceedings, SIGGRAPH 1999*, strony 269 – 276.

[Isenburg00] Isenburg Martin: „Triangle Strip Compression”, *Graphics Interface*, strony 197 – 204, 2000.

[Marselas02] Marselas Herb: „Optymalizacja dostarczania wierzchołków w OpenGL”, *Perelki programowania gier. Vademecum profesjonalisty. Tom 1*, Wydawnictwo Helion, 2002.

[Microsoft00] Microsoft DirectX 8.0 Software Development Kit, dostępne pod adresem <http://www.msdn.microsoft.com/downloads>, 2000.

[Neider99] Neider Jackie i inni: *OpenGL Programming Guide, Version 1.2*, Addison Wesley, 1999.

[Nvidia00] Nvidia NvTriStrip v1.1., dostępne pod adresem http://developer.nvidia.com/view.asp?IO=nvtristrip_v1_1, 2000.