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 czoło 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.

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

Spis treści

Rozdział 4.5

Tworzenie, optymalizacja i rendering pasków trójkątów

Carl S. Marshall, Intel Labs
Carl.S.Marshall@intel.com

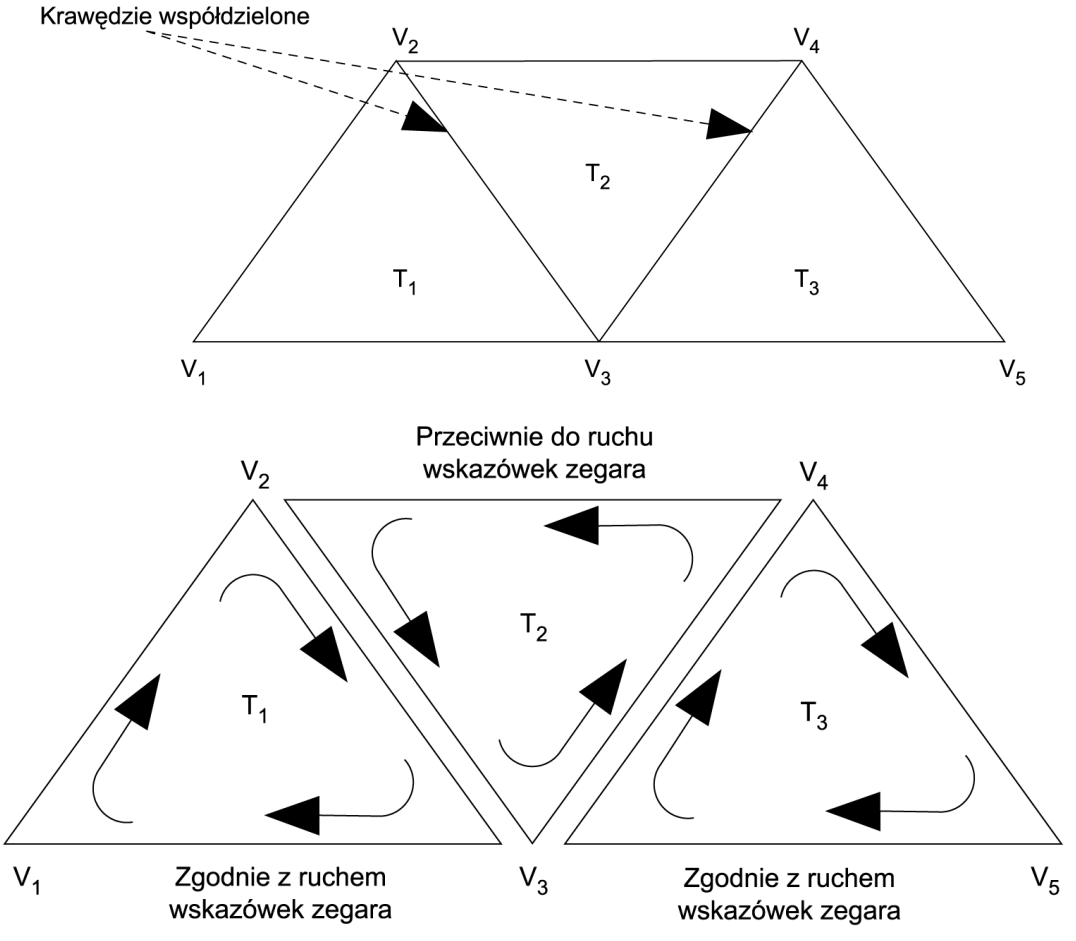
Obecnie, gdy istnieją wysokiej wydajności konsole do gier, paski trójkątów są stosowane coraz częściej do reprezentacji i renderingu geometrii. W tym rozdziale zajmę się zagadnieniem generowania pasków trójkątów z dowolnych modeli trójwymiarowych. Po omówieniu algorytmu pasków trójkątów przedstawię zalety 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 przesyłać 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 łat, 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.

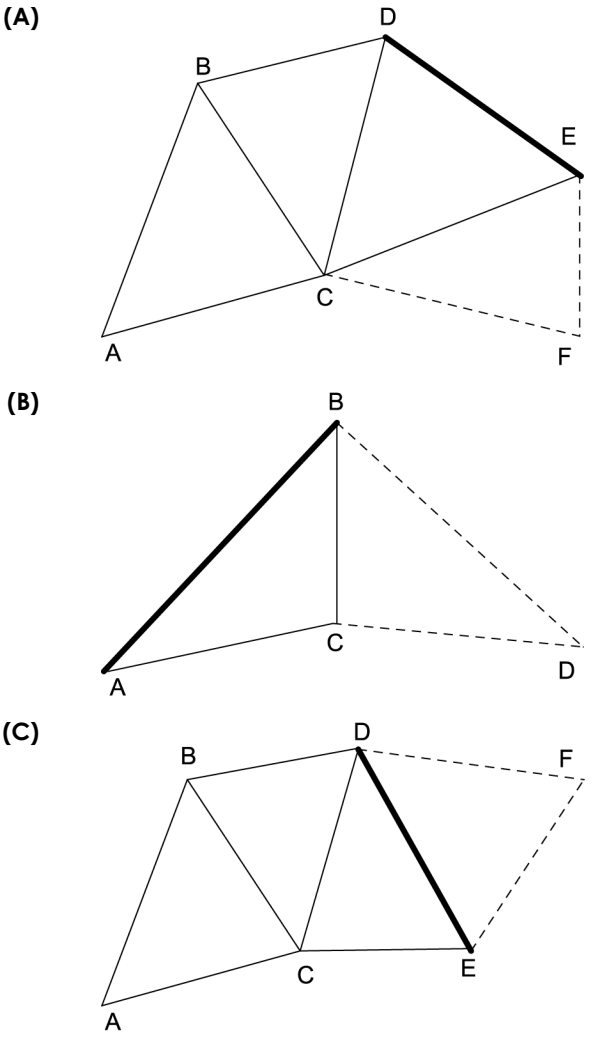
Ponieważ niemożliwe jest osiągnięcie idealnego paska trójkątów dla każdej siatki, należy wymyślić algorytm znajdowania najbardziej optymalnych pasków dla danego zestawu danych. Zastosowany algorytm generuje paski o jak największej długości dla dowolnego modelu trójwymiarowego złożonego z wielokątów. 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 przemienność 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.

$$C = (V_1+V_2+V_3)/3{,}0 \tag{4.5.1}$$

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 ograniczyłoby 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 położeń 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", *Perełki 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.