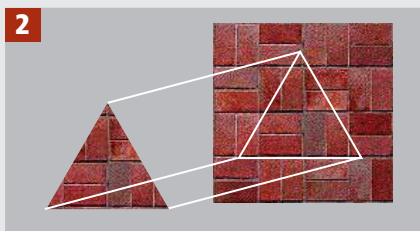
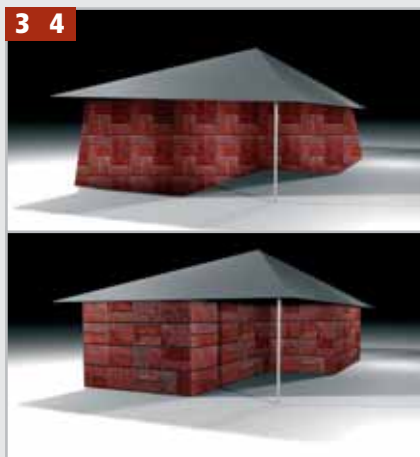


**Pierwszym etapem teksturowania jest przygotowanie odpowiedniej wielkości bitmap nanoszonych później na trójkąty o różnych wymiarach.**



Następnie z każdej bitmapy trzeba wyciąć odpowiedni, **pasujący do mapowanego wielokąta trójkąt**. Operacja ta nazywa się clippingiem.



Abym np. ściany w budynkach nie były powykrywane, akcelerator musi uwzględnić odpowiednią perspektywę. **Równie ważne jest nadanie płaskim powierzchniom chropowatości.**



Mając przygotowane wszystkie tekstury, karta przystępuje do ich nakładania. **Proces ten nazywa się mapowaniem.**

**Teksturowanie obciąża kartę graficzną w ok. 60%**

## Fotorealizm 3D

Kluczem do sukcesu podczas generowania grafiki 3D jest nie tylko szybkość karty, ale przede wszystkim technologie pomagające sprzętowo zrealizować na ekranie to, co wymyślili twórcy gier.

**Marcin Bieńkowski**

W poprzedniej części naszego cyklu o sprzętowym generowaniu grafiki (**CHIP 2/2005, s.86**) opowiedzieliśmy, jak współczesne karty graficzne tworzą złożony z trójkątów szkielet trójwymiarowej sceny. Teraz zajmijmy się tym, jak na ten stała nałożyć realistyczną powłokę i sprawić, żeby wszystkie występujące na scenie 3D przedmioty i postacie wyglądały tak, jak w rzeczywistym świecie. Proces ten nazywa się renderingiem, a jedną z jego najważniejszych części jest teksturowanie.

### Sztuka origami

Rendering realizowany przez wszystkie współczesne akceleratory 3D podzielić można na cztery główne fazy: teksturowanie, modyfikacje nałożonych tekstur, cieniowanie oraz dodawanie efektów „atmosferycznych”, do których zalicza się m.in. dym, mgłę czy różnego rodzaju rozlane plamy oleju.

Wszystkie wymienione operacje są dzisiaj wykonywane przez kartę graficzną, niemniej kości 3D całkiem niedawno „nauczyły się” modyfikować nałożone już na obiekty tekstury. Stało się to możliwe za sprawą jednostek Pixel Shader. Te moduły wykonawcze zadebiutowały wraz z kartami ATI Radeon 8500 oraz nVidia GeForce3. Ich obsługa znalazła się zaś w bibliotekach graficznych DirectX 8.0 i OpenGL 1.3.

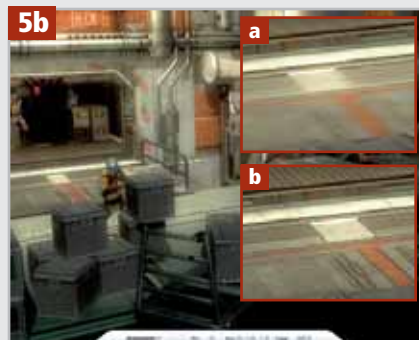
Wróćmy jednak do pierwszego i najważniejszego etapu renderingu – teksturowania. Faza ta zazwyczaj pochłania ponad 60% czasu potrzebnego na przygotowanie każdej sceny 3D. Sama tekstura to płaska mapa bitowa (bitmapa)

o z góry zadanej przez programistę wielkości. Najczęściej wykorzystuje się tekstury o rozmiarze 256x256, 512x512 oraz 1024x1024 piksele. Można rzec, że ta mapa bitowa po nałożeniu na elementy szkieletu sceny 3D jest cyfrowym odpowiednikiem spotykanych w realnym świecie powierzchni – np. udaje drewno lub skórę.

Sam proces nakładania tekstur, który nazywany jest mapowaniem, sprowadza się do „obwijania” brył teksturami. Mapowanie można porównać zatem do zawijania kanapki w folię aluminiową lub bardziej obrazowo: pakowania świątecznych prezentów w kolorowy papier. W tym miejscu warto też przypomnieć, że pojedynczy punkt tekstury nazywa się tekselem (od ang. texture element). Termin ten często jest błędnie stosowany na zmianę ze słowem piksel (ang. picture element). Mając już tę podstawową wiedzę, udajmy się naszą wędrówką po teksturowaniu. Zaczniemy ją od procesu przygotowania tekstur, jeszcze przed nałożeniem ich na szkielet sceny 3D.

### 1 MIP mapping, czyli jak sobie radzić z teksturami

Jak pamiętamy z pierwszej części artykułu o generowaniu grafiki 3D, gotowy szkielet sceny składa się z setek tysięcy trójkątów o różnej wielkości. Przygotowanie tekstur pasujących do każdego trójkąta znajdującego się na scenie nie jest możliwe, gdyż liczba potrzebnych bitmap musiałaby iść wówczas w setki milionów! Z takim ogromem wymienianych pomiędzy



Dwu- i trójliniowe filtrowanie tekstur (a) zapobiega m.in. efektowi tzw. pikseloży. Technika ta „skleja” też ze sobą mapy MIP różnych poziomów, tak aby nie były widoczne krawędzie przejść między nimi. Najlepsze efekty wizualne daje zaś filtrowanie anizotropowe (b).



Ostatnią fazą teksturowania są modyfikacje nałożonych już na obiekty bitmap. Akcelerator wykonuje je, **wykorzystując do tego celu jednostki Pixel Shader**. Otrzymany efekt końcowy zależy od fantazji programisty. Instrukcje shaderowe mogą np. rozkruszyć cegłę.

procesorem a kartą graficzną danych nie porażą sobie nawet najszybszy komputer.

Oczywiście najprostszym sposobem na ominięcie tego problemu jest przesłanie do pamięci karty kilku podstawowych wzorów tekstur, a następnie ich przeskalowanie, tak aby każda z nich pasowała pod względem rozmiaru do konkretnego użytego na scenie trójkąta. Niestety, taka operacja trwałaby strasznie długo (kilkanaście sekund na jedną klatkę animacji), nie mówiąc już o tym, że trzeba by było gdzieś te wszystkie tekstury zmieścić.

Aby nie wpędzić się w niepotrzebne obliczenia, już w latach pięćdziesiątych ubiegłego wieku wymyślono technologię MIP mappingu (Multum in Parvo – wiele w niewielu). Polega ona na utworzeniu z podstawowej tekstury kilku (zazwyczaj ośmiu) wzorców, które są następnie wykorzystywane do pokrywania elementów sceny. Wzorce te, nazywane poziomami mapy MIP, to nic innego jak przeskalowane bitmapy, z których każda jest czterokrotnie mniejsza od poprzedniej. Jeśli pierwsza miała rozmiar  $1024 \times 1024$  piksele, to następna będzie miała wielkość  $512 \times 512$  punktów, kolejna  $256 \times 256$  itd. Następnie do teksturowania wybiera się najbardziej zbliżone wielkością do mapowanego trójkąta mapy MIP – małe do małych, a duże do dużych trójkątów.

## 2 Clipping, czyli nożyczki w rękach akceleratora

Sama metoda MIP mappingu nie rozwiązuje jednak problemu dopasowania tekstur do trójkątów – wszak tekstury są kwadratowe. Jak zatem z kwadratu uzyskać trójkąt? Odpowiedź na to pytanie jest prosta – najlepiej „wyciąć” odpowiedniej wielkości trójkąt. Operacja wyodrębnienia trójkąta o żądanej wielkości z kwadratowej tekstury nazywa się clippingiem. Z punktu widzenia akceleratora polega ona na określeniu współrzędnych trzech punktów na teksturze, a następnie przyporządkowaniu ich (rozpięciu) trzem stosownym wierzchołkom teksturowanego trójkąta. Jest to czynność stosunkowo prosta i nie angażuje ona dużej mocy obliczeniowej układu. Przy clippingu i określaniu punktów zaczepienia należy jeszcze pamiętać o odpowiedniej orientacji trójkątnych fragmentów tekstur – zwłaszcza przy „obwijaniu” jedną teksturą wielu trójkątów należących do tej samej bryły, np. przy mapowaniu kuli. Wówczas znajdujące się obok powycinane trójkątne kawałki muszą do siebie idealnie pasować, tak aby wzór na teksturze tworzył zawsze jedną całość.

No dobrze, ale co zrobić, gdy kwadratowa tekstura okazuje się ciut za mała, żeby z niej wyciąć odpowiedni trójkąt, a innej mapy MIP nie można użyć (kolejne poziomy map MIP wykorzystywane są zawsze do teksturowania coraz to bardziej oddalających się od obserwatora obiektów)? W takiej sytuacji stosuje się dwa rozwiązania. Pierwsze polega na włożeniu tekstury w trójkąt i „dosztukowaniu” brakujących fragmentów, w podobny



**W teście Canyon Flight z 3DMarka 05 do nałożenia tekstur na ściany kanionu wykorzystano własną metodę teksturowania, napisaną wyłącznie w programie shaderowym.**

sposób jak robi to krawiec. Operacja ta realizowana jest zazwyczaj przez procesor komputera i ten algorytm wykorzystywany był do niedawna tylko w takich programach jak np. 3ds max. Obecnie takie sztukowanie realizuje się coraz częściej za pomocą programów dla Pixel Shaderów.

Drugą, prostą techniką „sztukowania” dziur jest próbkowanie punktowe (ang. point sampling texturing). Związane jest ono bezpośrednio ze sposobem nanoszenia tekstur, polegającym na kopiowaniu punkt po punkcie teksteli z bitmapy na mapowany trójkąt. Jeśli powierzchnia trójkąta jest większa od tekstury, to powiela się wówczas niektóre tekstele na kilka sąsiadujących ze sobą punktów trójkąta – można rzec, że „naciąga się na siłę” teksturę na wielokąt. Niestety, wadą próbkowania punktowego jest to, że powielanie teksteli objawia się efektem „pikselozy”. Jest on szczególnie wyraźny wtedy, gdy tekstura musi pokryć obszar znacznie większy niż ona sama – dzieje się tak głównie podczas mapowania małych, oddalonych od obserwatora obiektów. Teksteli widać wyraźnie, gdy bohater gry podejdzie do ściany. Wówczas dobrze wyglądające z daleka elementy zamieniają się w duże, rozmazane kwadraty. Dzisiejsze akceleratory zapobiegają takim nieciekawym efektom ubocznym dzięki stosowaniu różnych typów filtrowania tekstur, o czym za chwilę.

## 3 Korekcja perspektywy: sposób na krzywe tekstury

Podczas generowania sceny 3D wszystkie przetwarzane obiekty opisane są w trójwymiarowej przestrzeni. Później jednak będą one wyświetlane na płaskim ekranie monitora. I tu dochodzą kolejne kłopoty związane z dopasowywaniem tekstur. Otóż podczas odwzorowania dużych, płaskich powierzchni, takich jak np. ściany, sufity lub podłogi, które składają się ze sporej wielkości trójkątów ustawionych niemal prostopadle do płaszczyzny ekranu, pojawiają się problemy z zachowaniem odpowiedniej perspektywy. Elementy sceny „rozłazą się”, tworząc dziwnie powykrzywione wzory. Efekt ten wynika z przesunięcia się współrzędnych trójkątów względem punktów zaczepienia wyciętych w trójkąty tekstur. Jest to typowy błąd obliczeniowy związany z zaokrąglaniem kalkulacji.

## Kompresja tekstur – jak sobie radzić ze zbyt dużą ilością danych

Podczas generowania trójwymiarowej grafiki akcelerator wymienia z procesorem i pamięcią operacyjną olbrzymie ilości danych. Gros z nich stanowią tekstury – mogą one zająć prawie całą przepustowość magistrali AGP lub PCI Express. Tekstury wypełniają też niemal zupełnie pamięć karty graficznej. Nic więc dziwnego, że do zmniejszenia objętości bitmap opracowano algorytmy kompresji tekstur. Podobnie jak w wypadku plików JPEG, są to mechanizmy stratne, lecz mimo to dają wysoką jakość obrazu.

Najstarszy algorytm kompresji tekstur, wynaleziony przez firmę S3, to S3TC (S3 Texture Compression). Znany on jest również pod nazwą DXTC (DirectX Texture Compression), gdyż wykorzystuje się go standardowo w bibliotekach DirectX, począwszy od wersji 6.0. Maksymalny stopień upakowania dla DXTC wynosi 6:1.

Wraz z nową rodziną Radeonów z serii X800 firma ATI wprowadziła swój własny algorytm

pakowania tekstur – 3Dc. Jak twierdzi ATI, przy zachowaniu takiego samego współczynnika kompresji 4:1 bitmapy spakowane algorytmem 3Dc są dwa razy wyższej jakości niż w wypadku DXCT. Dotyczy to zwłaszcza tekstur z chropowatościami nałożonymi metodą mapowania przemieszczeń.



Opracowana przez firmę ATI metoda kompresji tekstur 3Dc ma według producenta kilkakrotnie poprawić jakość spakowanych stratnym algorytmem tekstur.

Każdy nowoczesny akcelerator w celu wyeliminowania tych niepożądanych efektów stosuje tzw. algorytmy korekcji perspektywy (ang. perspective correction). Metoda ta polega na stworzeniu na horyzoncie wirtualnego punktu, tak jak robią to malarze. Teraz każda linia poprowadzona z dowolnego miejsca sceny 3D musi zbiec się w wyznaczonym punkcie odniesienia. Następnie karta graficzna – zgodnie ze znaną już od XIII w. zasadą rzutu perspektywicznego – nakłada odpowiednio dopasowane, zorientowane i przycięte tekstury, ale już bez przestrzennych deformacji.

## 4 Bump mapping i mapowanie środowiskowe

Mimo nałożenia na szkielet sceny 3D nawet najlepszych tekstur nasz wirtualny świat ciągle będzie wyglądał dość sztucznie. Za to niekorzystne wrażenie odpowiadają gładkie, płaskie bitmapy. Jak zatem tworzy się złudzenie rzeczywistej, chropowatej struktury przedmiotów? Odpowiedź na to pytanie przynosi technika bump mappingu, w której wykorzystuje się dodatkową bitmapę nazywaną mapą wybojów (ang. bump maps).

Mapa wybojów to nic innego jak płaska, stworzona w odcieniach szarości bitmapa, gdzie „jasność” każdego punktu oznacza stopień jego „wypukłości”. Po złożeniu mapy wybojów z teksturą podstawową otrzymujemy zatem dodatkową informację o wzajemnym usytuowaniu każdego punktu w odniesieniu do płaszczyzny teksturowania. Przy obliczaniu oświetlenia teksele znajdujące się wyżej są jaśniejsze od tych położonych niżej. Dzięki temu cała oświetlona tekstura bardzo dobrze imituje chropowatości i wgłębienia powierzchni.

Przedstawiona powyżej najprostsza metoda emulowania wypukłości nosi nazwę wstępnego przeliczanego bump mappingu (ang. pre-calculated bump mapping). Pozwala ona dość wiernie i przede wszystkim przy wykorzystaniu niewielkiej mocy obliczeniowej odwzorować chropowatości nieruchomych i prostopadłych względem obserwatora powierzchni.

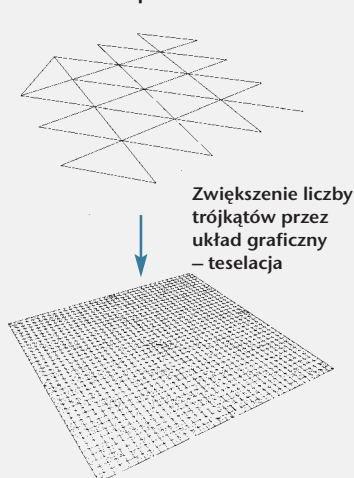
Jeśli przedmioty widać pod niewielkim kątem lub tekstury nakłada się na poruszające się przedmioty, wówczas stosuje się drugą z metod bump mappingu – tzw. tłoczenie wybojów (ang. emboss bump mapping). Odbija się ona w trzech etapach. W pierwszej fazie z mapy opisującej wygląd wypukłości (mapy wybojów) akcelerator tworzy dwa monochromatyczne obrazy. Jeden wykorzystywany jest do przedstawienia jaśniejszych obszarów, a drugi odpowiada za fragmenty zacienione. W drugim kroku obie bitmapy przesuwane są o kilka pikseli względem siebie (jedna do tyłu, druga do przodu) wzdłuż kierunku padającego światła. Ostatnią fazą tłoczenia jest złączenie rozsuniętych bitmap i połączenie ich z podstawową teksturą obiektu. Do „klejenia” obu map wypukłości używa się technologii alpha-blendingu, o której za chwilę.

Trzecią, jeszcze bardziej zaawansowaną metodą bump mappingu jest mapowanie środowiskowe (ang. environment mapping). Na początku zacznijmy od wyjaśnienia, co to jest mapa środowiskowa (ang. environment map). Jest to nic innego, jak tekstura zawierająca w sobie informacje m.in. o refleksach na powierzchni przedmiotu. Pochodzą one od rozstawionych na scenie białych i wielokolorowych źródeł światła, odbić od zwierciadeł i innych gładkich, wypolerowanych przedmiotów. Proces mapowania środowiskowego używany jest do uzyskania w grach wszelkiego rodzaju efektów luster, szklanych drzwi czy np. obrazów przedmiotów odbijających się na metalizowanej lub lakierowanej powierzchni.

Odmianą mapowania środowiskowego są techniki kubicznego, sferycznego i podwójnie parabolicznego mapowania środowiskowego (cube, spherical i dual paraboloid environmental mapping). Polegają one na nakładaniu na bryłę nie jednej, ale przynajmniej sześciu (dla kubicznego), ośmiu (dla parabolicznego) lub dowolnej, określonej przez programistę liczby (dla mapowania sferycznego) bitmap odpowiadających co najmniej obrazowi

## Mapowanie przemieszczeń

Siatka podstawowa

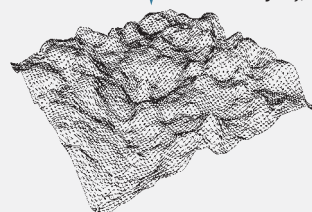


Zwiększenie liczby trójkątów przez układ graficzny – teselacja



Czterokilobajtowa bitmapa odkształceń (64×64×8 bitów)

Odształcenie siatki zgodnie z mapą przemieszczeń (jasne niżej, ciemne wyżej)



Nakładana tekstura

Klasyczny rendering – teksturowanie, cieniowanie itp.



Obraz końcowy

Technika Displacement Mapping (mapowanie przemieszczeń) wykorzystująca Vertex i Pixel Shadery nie emuluje, lecz rzeczywiście odtwarza chropowatości.

dolnej, górnej i czterech bocznych części otoczenia obiektu. Te przestrzenne mapowania środowiskowe zapobiegają sytuacji, w której tekstury zaczynają „odbijać” nie ten fragment sceny 3D, co trzeba, w chwili gdy obserwator się przemieści lub przedmiot ulegnie silnemu zniekształceniu.

Wspomnianą przed chwilą technikę łączącą mapowanie środowiskowe z bump mappingiem określa się nazwą EMBM (Environment-Mapped Bump Mapping). Pozwala ona otrzymać bardzo zaawansowane efekty wizualne, takie jak falująca powierzchnia wody, w której mogą się odbijać obrazy znajdujących się na brzegu



## Tekstury wolumetryczne – trzeci wymiar ściany

Wraz z bibliotekami DirectX 8.0 i OpenGL 1.2 pojawiła się możliwość wykorzystania zupełnie nowego rodzaju tekstur, a mianowicie tekstur wolumetrycznych. Te nowe bitmapy nie są już twórami płaskimi, lecz niosą ze sobą trójwymiarową informację – dane o wewnętrznej strukturze przedmiotu. Tekstury wolumetryczne, nazywane też warstwowymi, pojawiły się najpierw w medycynie, przy okazji rozpowszechnienia się takich technik diagnostycznych, jak tomografia komputerowa. Tutaj podczas prześwietlania pacjenta tworzona jest trójwymiarowa mapa składająca się z wielu przekrojów prześwietlanego organu. Po badaniu, gdy lekarz zechce obejrzeć wewnętrzny fragment tkanki, wystarczy, że komputer, modelując obraz, zdejmie niepotrzebne warstwy tekstury, nadając im zerową wartość kanału przezroczystości alpha. Dzięki temu łatwo zajrzeć do interesującego nas wewnętrznego obszaru narządu.

W wypadku gier wykorzystanie tekstur wolumetrycznych oznacza, że bitmapy przestają wyłącznie okrywać powierzchnię przedmiotów i zjawisk, ale wnikają w ich głąb. Jakie to niesie ze sobą konsekwencje? Po pierwsze, bardzo łatwo modelować wszelkiego rodzaju uszkodzenia. Załóżmy, że w grze ukruszył się kawałek ceglanej, otynkowanej ściany. W klasycznej metodzie generowania grafiki 3D, aby odwzorować „ukruszenie”, trzeba poświęcić na to znaczną moc procesora. Należy bowiem policzyć na nowo szkielet ściany, łącznie ze składającymi się z setek małych trójkątów uszkodzeniami i poszarpaniami. Następnie każdy najdrobniejszy uszkodzony detal musi być pokryty odpowiednią teksturą. Przy zastosowaniu tekstury wolumetrycznej proces obliczeniowy sprowadzi się do nadania części tekstele atrybutu przezroczystości. Wówczas odsłonią one wnętrze i będzie można zobaczyć najdrobniejsze detale odprysku, gdyż budowa i faktura ściany zawarte są w mapie wolumetrycznej. Równie często tekstury warstwowe wykorzystuje się do przestrzennej symulacji ognia – tutaj realistyczne ruchy płomienia bardzo



**Demo Vulcan firmy nVidia pokazuje, do czego można wykorzystać tekstury wolumetryczne w grach. Objętościowa bitmapa przyczepiona do szkieletu postaci z powodzeniem udaje nie tylko płomień, ale i dym.**

łatwo uzyskać, zmieniając cyklicznie jedynie atrybuty przezroczystości tekstury.

Tekstury wolumetryczne mają jednak istotną wadę – ich objętość jest wyjątkowo duża. Aby się zorientować, jak znaczne są rozmiary map wolumetrycznych, wystarczy zauważyć, że 32-bitowa dwuwymiarowa tekstura o wymiarach 16×16 pikseli ma objętość 1024 bajtów. Po dodaniu trzeciego wymiaru (16×16×16 punktów) rozrasta się ona już do 16 384 bajtów.

przedmiotów, osób itp. EMBM posłużył także do tworzenia w wielu grach kałuż na jezdni czy błyszczącej powierzchni rozgrzanego w słońcu asfaltu. Uzyskanie takich efektów możliwe jest dzięki nałożeniu na jeden teksele trzech bitmap: podstawowej tekstury, mapy wypukłości oraz mapy środowiska. Co ciekawe, technika EMBM jest starsza, niż mogłoby się wydawać – pojawiła się wraz z DirectX 6.0.

W bibliotekach DirectX 7.0 zaszyto zaś czwartą ze stosowanych obecnie metod odwzorowania wypukłości. Łączy ona szybkość obliczeniową tłoczenia wybojów z dokładnością techniki EMBM. Metoda ta to – nie wiedzieć czemu rzadko stosowana – technologia Dot3. W tym przypadku dla każdego wielokąta sceny 3D obliczany jest kąt odbicia padającego na niego światła lub wypadkowy wektor zawierający zbiorcze informacje o refleksach z wielu źródeł. Następnie otrzymaną wartość wektora modyfikuje się zgodnie z danymi zapisanymi w tzw. mapie odbić, w której umieszczono informacje o połyskujących fragmentach powierzchni. Na końcu otrzymaną bitmapę Dot3 łączy się z właściwą teksturą obiektu.

Ostatnią metodą bump mappingu jest opracowane trzy lata temu przez firmę Matrox mapowanie przemieszczeń, czyli displacement mapping. Technika ta polega na nadaniu płaskim obiektom prawdziwych, a nie – jak do tej pory – symulowanych wypukłości (patrz: s74). Fragment bryły, który chcemy zmodyfikować, dzielony jest na mniejsze trójkąty. Następnie jednostka Vertex Shader modyfikuje (wygina) powierzchnie bryły zgodnie z mapą przemieszczeń (odpowiednik mapy wybojów). Na końcu na taki „zwichrowany” fragment bryły nakłada się podstawową teksturę, którą wygina się zgodnie z zadanyymi uprzednio „zagnieceniami”. Z mapowaniem przemieszczeń radzą sobie karty zgodne z bibliotekami DirectX 8.1 oraz 9.0.

## 5 Scena prawie gotowa: teksturowanie i filtrowanie

Gdy mamy już wycięte tekstury, poprawioną perspektywę oraz przygotowane wszelkiego rodzaju mapy wypukłości i przemieszczeń, można przejść do zasadniczego teksturowania. W większości wypadków na jeden obiekt 76»

## Czym się różnią jednostki Vertex i Pixel Shader w wersji 2.0 i 3.0

Wraz z pojawieniem się kart graficznych zbudowanych na bazie nowej generacji układów nVidii z serii GeForce 6200, 6600 i 6800 na rynku zadebiutowała trzecia odsłona technologii Vertex i Pixel Shader. Wprowadzone przez nVidię zmiany spowodowały przede wszystkim przyspieszenie oraz zoptymalizowanie obliczeń w obu typach jednostek wykonawczych. Innymi słowy: karty z Vertex i Pixel Shaderami 3.0 znacznie szybciej (nawet do dwóch razy) przetwarzają programy modyfikujące szkielet sceny, oświetlenie oraz tekstury.

Dzięki wprowadzeniu nowych instrukcji (w tym pętli) oraz zwiększeniu liczby zmiennych i wydłużeniu dopuszczalnej długości programów Vertex i Pixel Shader 3.0 znacznie łatwiej się programuje niż jednostki 2.0. Należy jednak zaznaczyć, że otrzymywane w obu wersjach efekty wizualne powinny być zbliżone. Jedyną różnicą jest to, że aby otrzymać ten sam efekt w shaderach 2.0, programista musi się bardziej „namęczyć”, łącząc ze sobą kilka lub kilkanaście programów. W nowej odsłonie shaderów 3.0 wystarczy zaś jedna znacznie mniej skomplikowana aplikacja.

## Porównanie Vertex i Pixel Shaderów w wersji 2.0 i 3.0

	DirectX 9.0 – Shader Model 2.0	DirectX 9.0c – Shader Model 3.0
<b>Vertex Shader</b>		
Maks. liczba instrukcji	1024	65 536
Liczba rejestrów	12	16
Maks. liczba zagnieżdżeń pętli	4	256
<b>Pixel Shader</b>		
Maks. liczba instrukcji	64	1024
Liczba rejestrów	12	32
Szerokość instrukcji	96 bitów	128 bitów
Maksymalna liczba stałych	32	65 536

nakładanych jest co najmniej kilka różnych bit-map. Oczywiście są one mapowane równolegle w wielu jednostkach wykonawczych karty. Akcelerator nie czeka też, aż będzie miał do dyspozycji wszystkie gotowe tekstury, proces teksturowania rozpoczyna od razu od takich elementów sceny, które są już zgromadzone w pamięci, stopniowo dodając później przygotowywane bit-mapy, np. mapy wypukłości.

Z procesem wypełniania sceny teksturami związany jest też wspomniany już alpha-blending. Służy on do „zarządzania” przezroczystościami i dzięki temu ułatwia nakładanie jednej tekstury na drugą. Technika alpha-blendingu wykorzystuje tzw. kanał alpha, określający stopień przezroczystości nakładanej tekstury. Przyjmuje on wartości od zera (obiekt zupełnie transparentny) do 255 (powierzchnia nieprzezroczysta). Wartości pośrednie oznaczają zaś odpowiedni stopień przezroczystości tekstury. Wartość kanału alfa ukryta jest w czterech parametrach opisujących punkt bitmapy. W wypadku korzystania z 32-bitowego koloru 24 bity opisują barwy RGB, a brakujące osiem bitów to właśnie kanał alfa.

W trakcie teksturowania akcelerator wykonuje jeszcze jedną czynność, a mianowicie wymienione wcześniej filtrowanie tekstur. Ma ono zapobiec zarówno efektowi „pikselozy”, jak i rozmyć niekiedy zbyt wyraźne granice pomiędzy trójkątami, na których nałożono dwa kolejne poziomy map MIP (np. przy naprzemiennym wypełnianiu trójkątów sąsiednimi poziomami map MIP). Filtrowanie przeciwdziała też „gubieniu” detali związanemu ze skalowaniem map MIP – szczegóły, takie jak np. spójnienia cegieł, mogą stać się niewidoczne, gdyż karta „nie wie”, jakie punkty na teksturze są istotne.

Najprostsza i najstarsza technika filtrowania tekstur to tzw. filtrowanie dwuliniowe (ang. bilinear filtering). Metoda ta polega na przyporządkowaniu każdemu filtrowanemu tekselewi koloru otrzymanego w wyniku interpolacji czterech sąsiednich punktów tekstury. Jej wadą jest to, że na przejściach pomiędzy poszczególnymi mapami MIP wciąż widoczne są krawędzie. Bardziej zaawansowanym sposobem jest filtrowanie trójliniowe (ang. trilinear filtering). Tutaj do ujednolicenia barwy oddalonych obiektów

używa się dwóch kolejnych map MIP. Najpierw metodą dwuliniową interpoluje się barwy wynikające z pierwszej mapy MIP, później zaś z drugiej. Uśredniony wynik uzyskuje się zatem na podstawie wartości ośmiu teksteli.

Kolejną, bardziej skomplikowaną metodą filtrowania tekstur jest powszechnie dziś stosowane filtrowanie anizotropowe (ang. anisotropic filtering). Tutaj przy uśrednianiu koloru bierze się pod uwagę orientację przestrzenną teksteli względem obserwatora. Dzięki temu obszary, z których interpolowane są brakujące wartości, układają się w kształt elipsy, prostokąta lub rombu (są to tzw. figury anizotropowe), gdzie długa oś figury wyznacza kierunek procesu filtrowania. Do metody filtrowania anizotropowego wykorzystuje się obecnie zazwyczaj od ośmiu (poziom filtra 1x) do 128 teksteli (16x).

## 6 Czas na Pixel Shader, czyli operacje na pikselach

Po przeprowadzeniu filtrowania tekstur do akcji przystępują moduły Pixel Shader, które są w stanie zmodyfikować nałożone już na obiekty bit-mapy. Co ciekawe, programy obsługujące shadera mogą w niemal dowolny sposób łączyć ze sobą tekstury składowe. Dzięki temu sposób mapowania zależeć może wyłącznie od wyobraźni i umiejętności programisty. Można więc pominąć wszystkie wymienione wcześniej metody teksturowania i zaimplementować swoją własną, która da niespotykane dotąd realistyczne efekty.

Niestety, programiści piszący gry nie wykorzystują w pełni potencjału, jaki niosą ze sobą programy shaderowe, gdyż takie programowanie nie jest proste. Łatwiej też skorzystać z gotowych, sprawdzonych procedur. Niemniej teksturowanie shaderowe i ewentualne modyfikowane nałożonych już obiektów daje niesamowite efekty. Wyłącznie na shaderach oprogramowano tekstury ścian wąwozu w najnowszym teście Canyon Flight pochodzącym z benchmarku 3DMark05. Powierzchnia skał wąwozu powstała dzięki programowi, którego kod osiągnął limit 64 instrukcji przewidziany w specyfikacji Pixel Shader 2.0. Skały otrzymano, łącząc dwie mapy kolorów, dwie mapy normalne i dodając do tego rozproszone cieniowanie metodą Lamberta.

I tu właśnie dotarliśmy do miejsca, w którym mamy gotową, oteksturowaną scenę 3D. Teraz należałoby ją oświetlić. Ale o tym opowiemy w kolejnej części cyklu o generowaniu trójwymiarowej grafiki. K



**Zastosowanie programów shaderowych do generowania grafiki 3D przynosi znaczną poprawę jakości wyświetlanej grafiki. Po lewej scena 3D (gra FarCry) z wyłączoną jednostką Pixel Shader, po prawej zaś to samo ujęcie po włączeniu jednostki Pixel Shader 3.0 (nVidia GeForce 6800).**

## Więcej informacji

### Generowanie grafiki 3D

<http://www.guru3d.com/>  
<http://www.beyond3d.com/>

### Literatura

„GPU Gems – Programming Techniques, Tips, and Tricks for Real-Time Graphics”, Edited by Randima Fernando, nVidia & Addison-Wesley, Boston 2004.