

Wprowadzenie do GPGPU

Pracownia Gier w OpenGL

Michał Drobot
Drobot.org

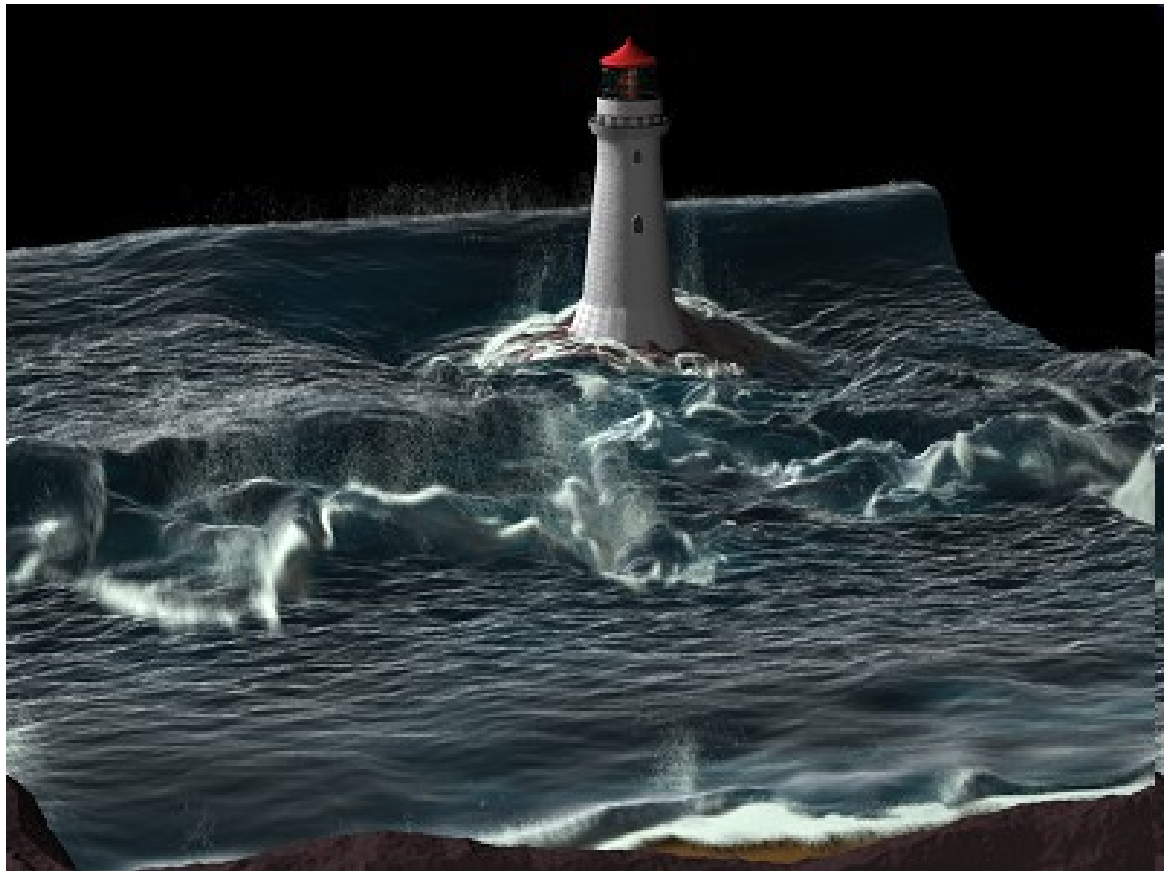
Cel

Nowe GPGPU

Klasyczne GPGPU

Mapowanie

Optymalizacje



Plan

- GPGPU
- Istniejące platformy
 - Dedykowane
 - Klasyczne
- Sposób mapowania algorytmu
 - Mapowanie danych
 - Mapowanie obliczeń
- Optymalizacje sprzętowe
 - Stencil / Z - buffer
 - Sampler states
- Typowe zastosowania

GP GPU

■ GP GPU

- Obliczenia ogólnego zastosowania wykonywane na karcie graficznej
- Duże znaczenie naukowe ze względu na łatwość mapowania niektórych algorytmów na architektury wielopotokowe
 - Olbrzymia wydajność GPU w zastosowaniach wektorowych
- Przyspieszanie standardowych obliczeń w grach będących dotąd domeną CPU
 - Fizyka
 - AI
- Powiązane ideowo i wykonawczo z postprocessingiem graficznym

Istniejące platformy

- Platformy dedykowane GPGPU
 - GPU zgodne z DX 10.0
 - Nvidia G80
 - Nvidia CUDA
 - Nvidia TESLA
 - Ati HD
 - Ati STREAM
 - Ati CTM

Istniejące platformy

- Oferują
 - Wsparcie programistyczne dla GPU
 - Działają jak wrappery
 - Mapują typowe dla CPU operacje na GPU bez udziału użytkownika
 - Posiadają duży narzut obliczeniowy
 - Umożliwiają prace w standardowym C / C++ przy akceleracji sprzętowej
 - Dają bezpośredni dostęp do sprzętu
 - Obecnie mimo narzutu są dobrą alternatywą dla CPU z racji olbrzymiej wydajności ALU (większość mapowanych algorytmów 40:1)

Istniejące platformy

- GPU zgodne z DX 11.0
 - DX 11 wprowadza SM 5.0
 - Computational Shader
 - Tryb obsługi shaderów ukierunkowany na obliczenia ogólnego zastosowania
 - Polimorfizm, klasowość i inne cechy C++ dla shaderów
 - Wykonywane sprzętowo z małym narzutem obliczeniowym

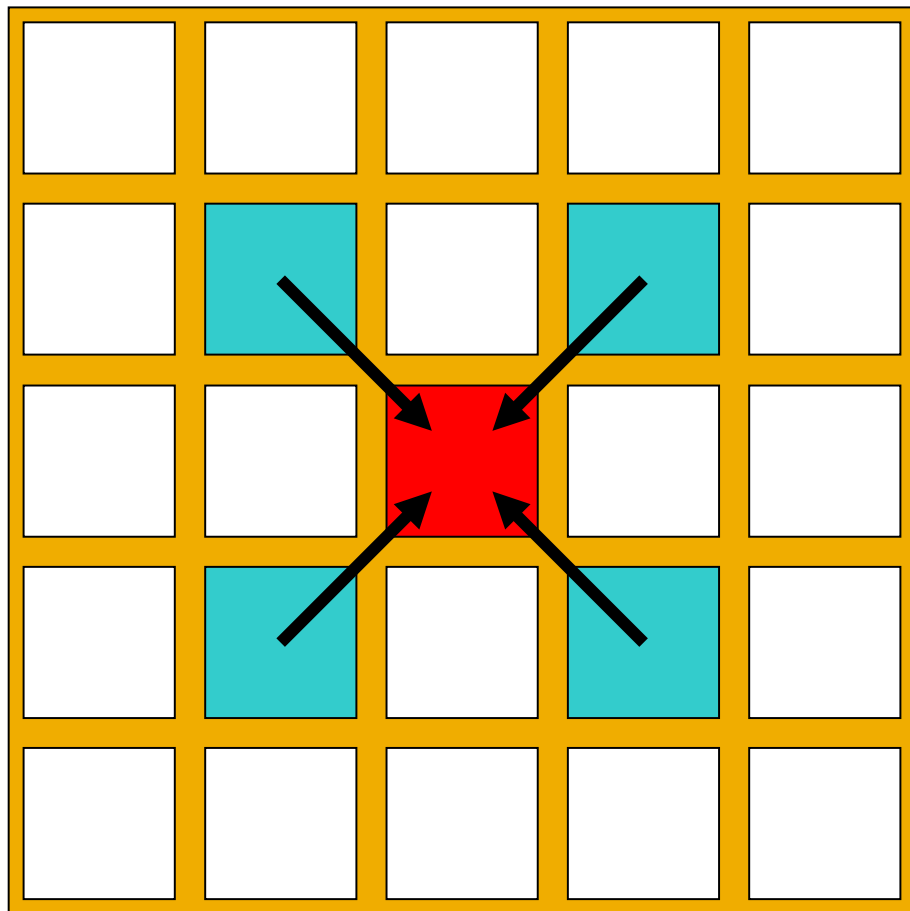
Istniejące platformy

- „Klasyczne GPGPU”
 - Wymusza mapowanie algorytmów na platformę graficzną nieprzystosowaną do obliczeń ogólnego zastosowania
 - GPGPU wykonywane na X360 i PS3 wymaga podejścia klasycznego
 - Większość użytkowników posiada GPU nie wspierające nowego GPGPU
 - Większa wydajność niż przy użyciu wrappera

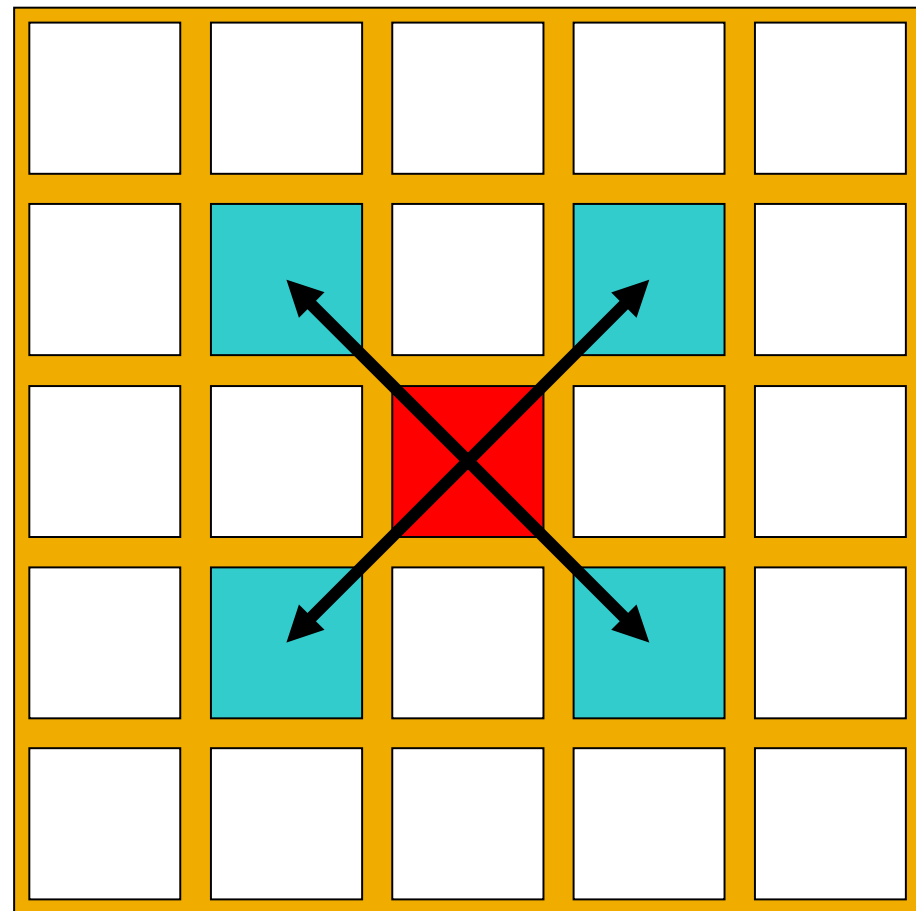
Sposób mapowania algorytmu

- Ogólny pipeline
 - Załadować zmienne do tekstur
 - tekstury GPU == tablice CPU
 - Koordynaty tekstur == domeny obliczeniowe
 - Renderuj czworokąt
 - Wykonaj obliczenia w shaderach
 - Pixel Shader == pętle zewnętrzne CPU
 - Sampler tekstur == odczyt pamięci CPU
 - Zapisz wynik obliczeń (renderingu) jako pixele w czworokącie
 - Render To Texture == zapis pamięci CPU
 - koordynaty vertexów == zakres obliczeń

Sposób mapowania algorytmu



Gather



Scatter

Sposób mapowania algorytmu

■ Metody pobierania danych

■ Gather

- Dla wyniku danej komórki zbieramy dane z innych niezależnych komórek
- Niezależny odczyt
 - $A = x[i]$
- Realizowane przez texture sampler
 - `texture2D()`

Sposób mapowania algorytmu

- Metody pobierania danych
 - Scatter
 - Dla danych komórek rozprzestrzeniaamy wyniki w inne zadane niezależne komórki
 - Niezależny zapis
 - $A[i] = x$
 - Niewykonalne na GPU

Sposób mapowania algorytmu

- Korzystamy z algorytmów typu Gather
- Algorytmu typu Scatter czasem możemy 'odwrócić'
- Rozwiązania projektujemy myśląc w trybie Gather
- Nie istnieje możliwość jednoczesnego odczytu i zapisu => brak możliwości zastosowania algorytmów dynamicznych w pojedynczym przebiegu

Sposób mapowania algorytmu

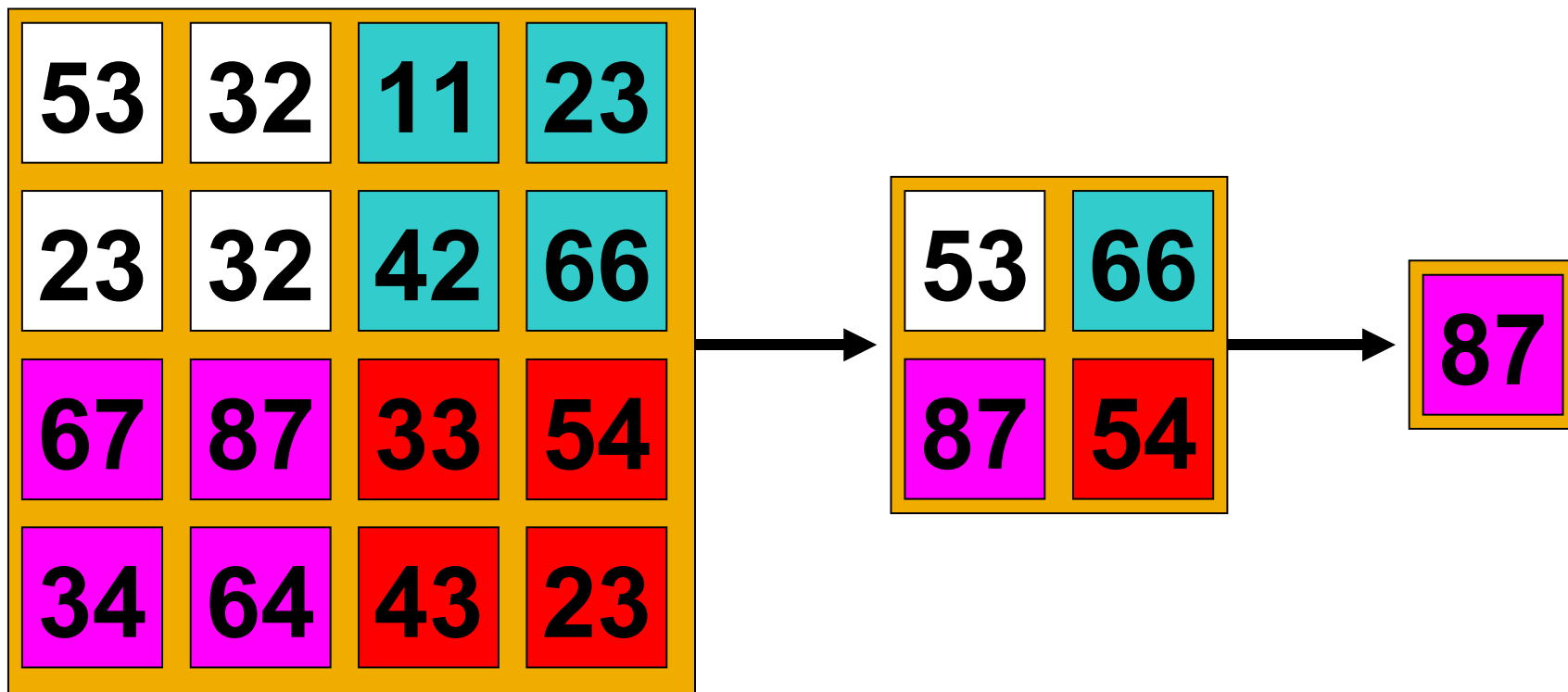
- PS wykonuje określoną funkcję na każdym pixelu (bloku danych) równoległe
- Przechowujemy dane wejściowe jako texele w teksturze
- Rysujemy czworokąt którego pixele stanowią wynik konkretnego działania na texelach textury wejściowej
- Zapisujemy wyrenderowany czworokąt jako texture do dalszych obliczeń

Sposób mapowania algorytmu

- Łatwość przeprowadzenia operacji złożonych
 - Min, max , sumy, średnie itp..
- Wiele przejść podczas obliczeń
 - Ciągi : textura -> RTT -> textura -> RTT...
 - Zawężanie zbioru danych
 - Ciągi renderujące do mniejszych tekstur
 - Skalowalność ‘lokalności’ pojedynczego przejścia jest związana z ilością przejść
 - Więcej odczytów pamięci zwiększa czas przejścia ale zmniejsza ilość wymaganych przejść

Sposób mapowania algorytmu

- Przykład : max - 1:4



Sposób mapowania algorytmu

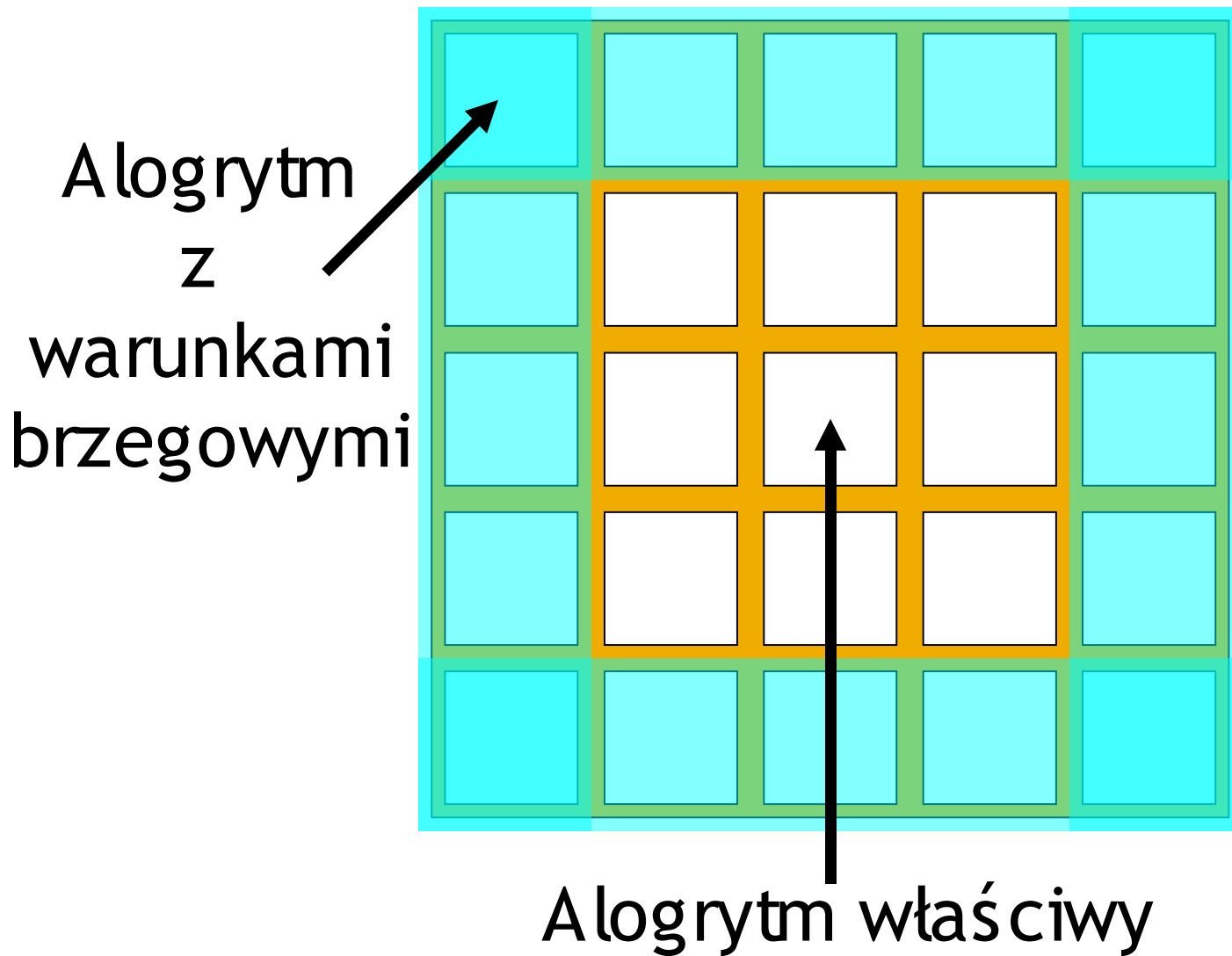
■ Branching

- W GPGPU należy szczególnie uważać na flow control
 - Często mapowane algorytmy ciężko przygotować do działania równoległego
 - Stosowanie Flow control może okazać się nieefektywne
 - Kompilatory preferują Predicated Branching
 - Wykonanie obu 'gałęzi'
 - Wybranie wyniku z jednej
 - Wiąże się to z wielokrotnym liczeniem

Sposób mapowania algorytmu

- Zamiast stosować branching czasem opłaca się dokonać określonych przebiegów PS dla ograniczonego zbioru pixeli
 - Stosowane w podczas liczenia warunków granicznych wielu symulacji ciągłych

Sposób mapowania algorytmu



Optymalizacje sprzętowe

- Wykorzystanie Z-Culling do przyspieszenia obliczeń
 - Zapisujemy do Z-buffera wartość 0 jeśli chcemy pominąć obliczenia
 - Zapisujemy wartość 1 jeśli chcemy wykonać obliczenia
 - PS dajemy na wejściu wartość głębi 0.5
 - W ten sposób obliczenia zostaną pominięte sprzętowo
 - Przydatne np. w nieruchomych obszarach symulacji
 - Podobnie można wykorzystać Stencil Buffer
 - Różne implementacje Hi-Z nie gwarantują pominięcia obliczeń co do pixela

Optymalizacje sprzętowe

- S tałe dla systemu obliczeniowego liczymy na CPU i przekazujemy do GPU
- W zależności od konfiguracji sprzętowej funkcje wydajniejsze może być przechowywanie nisko wymiarowych funkcji w teksturach
 - Odczyt interpolowany bądź punktowy w zależności od funkcji

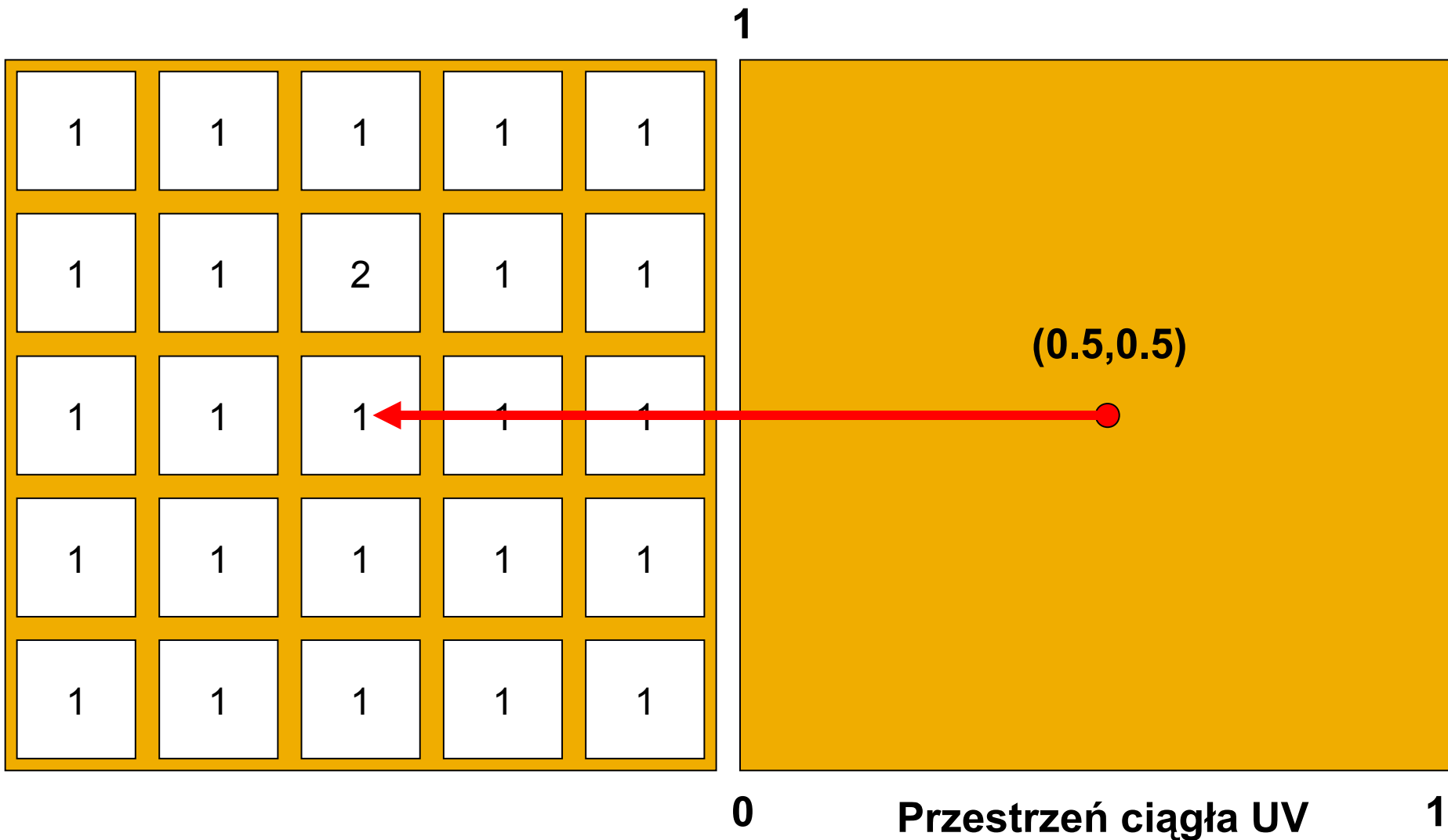
Optymalizacje sprzętowe

- Mniej bitowe struktury danych są szybsze w obliczeniach
 - Podstawowa jednostka obliczeniowa : float
 - Obsługa F32 , F64, I32 , I64
 - Wykorzystujemy istniejące formaty w sposób ekonomiczny R16G16, RGBA64 , R16G16B16A16F
 - Brak prostego wsparcia struktur danych
 - Istnieją liczne implementacje hashmap, drzew, systemów kubełkowych etc.

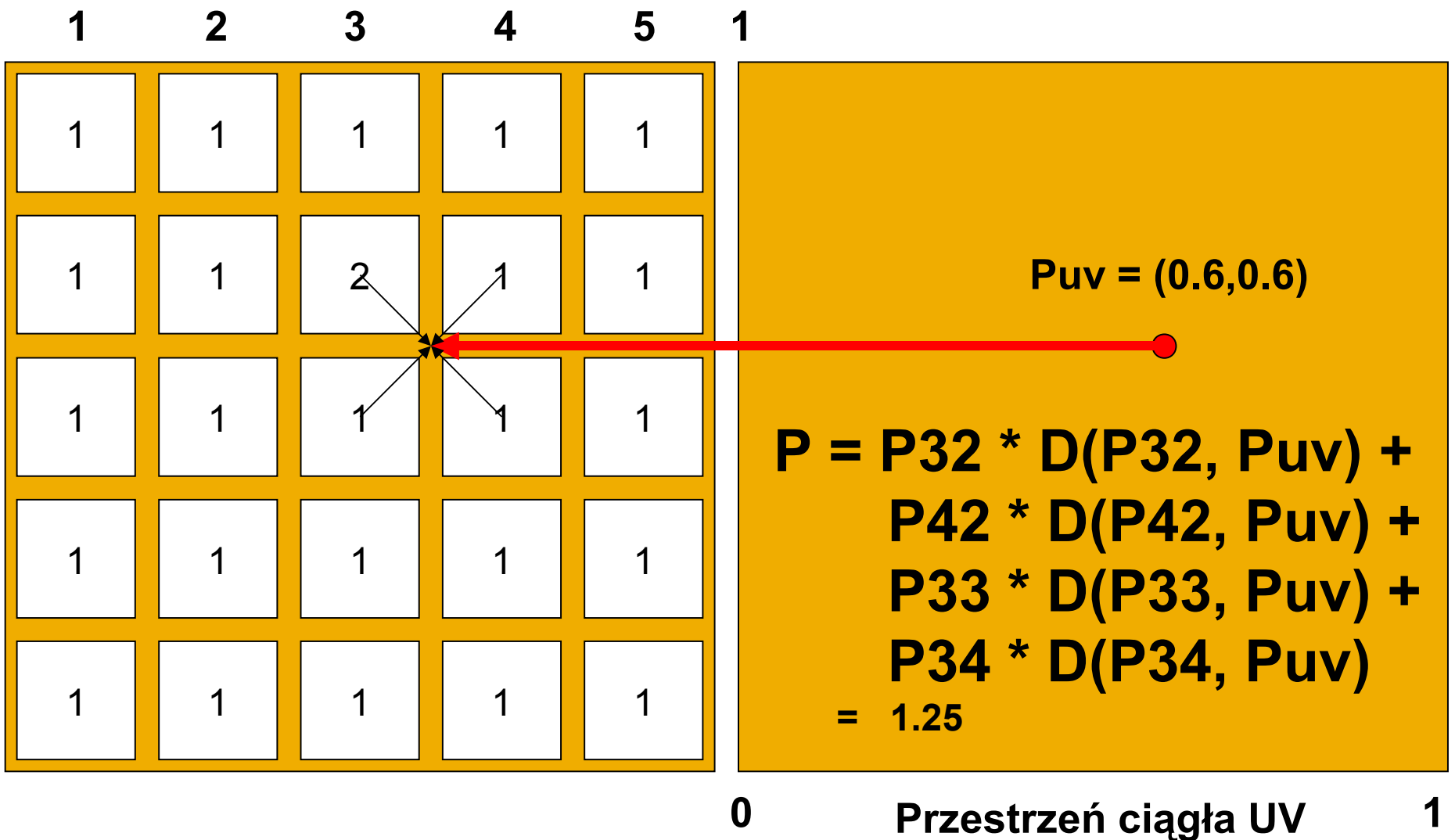
Optymalizacje sprzętowe

- Wykorzystanie stanów samplera
 - Sampler textury może pracować w trybie
 - POINT - sampluje wg. Nearest neighbour
 - LINEAR - podaje wartość wg interpolacji bilinearnej 2x2 texeli
 - W sytuacjach gdy liczymy wartości uśrednione możemy wykorzystać sprzętowe wsparcie LINEAR samplera
 - Przydatne również w wielu metodach relaksujących
 - Smoothing symulacji itp..

Optymalizacje s prętowe



Optymalizacje sprzętowe



Typowe zastosowania

- Aplikacje o dużym stosunku obliczeń do przesyłu danych
- Równania różniczkowe
- Automaty komórkowe
- Algebra liniowa
- Systemy cząsteczkowe
- Przetwarzanie danych
- Wspomaganie istniejących aplikacji
 - Photoshop
 - Dekodowanie video

Typowe zastosowania

- Zastosowania w grach
 - Detekcja kolizji
 - Fizyka
 - Ciał stałych
 - Cząsteczek, dymu, wody , ciał miękkich
 - Systemy cząsteczkowe
 - Raycasting / Raytracing
 - Systemy komórkowe AI

Typowe zastosowania

- Istniejące systemy fizyczne wykorzystują obliczenia GPGPU
 - Havok FX
 - Nvidia Physix

Pytania?



Plan wykładów

- Architektura oraz optymalizacja shaderów
- Modelowanie właściwości fizycznych materiałów
- Obliczenia na teksturach / tablicach
 - Wprowadzenie - obsługa render targetów
 - Post processing obrazu - metody i praktyki
 - GPGPU - obliczenia ogólnego zastosowania na GPU

Punktacja etapu 'rozbudowy'

- Do zdobycia 10 (+2) pkt
 - 5 pkt
 - Wybranie 3 materiałów a następnie wymodelowanie ich oraz zintegrowanie z frameworkiem
 - Wybierane z puli materiałów
 - 2 proste , 1 średni
 - Dodatkowo do +1 pkt za zaawansowany materiał
 - 5 pkt
 - Integracja modułu obsługi render targetów
 - Wprowadzenie 3 efektów post processingu
 - Wybierane z puli efektów
 - 2 proste, 1 średni
 - Dodatkowo do +1 pkt za zaawansowany efekt

Punktacja etapu 'rozbudowy'

- S hadery będą oceniane pod względem
 - Przejrzystości
 - Wydajności (optymalizacji)
 - Prawidłowego efektu
 - Zastosowanej metody
 - Pomysłowości
- Zastrzegam możliwość rozmowy nt. konkretnego rozwiązania, jego zasadności oraz zrozumienia zagadnienia

Dziękuję za uwagę

Michał Drobot
Drobot.org