Zachodniopomorski Uniwersytet Technologiczny w Szczecinie

Wydział Informatyki

Wprowadzenie do programowania shaderów

Gry komputerowe

15.05.2019

mgr inż. Marek Wernikowski

Wczytywanie shaderów

Potok graficzny

Zmienne w shaderach

Atrybuty

Uniformy

Clip Space

Tworzenie kształtów

Face Culling

Rasteryzacja

Test głębokości

Efekt komiksowy

Wczytywanie shaderów

```
m_program = new QOpenGLShaderProgram;
m_program->addShaderFromSourceFile(QOpenGLShader::Vertex, "resources/shader.vs");
m_program->addShaderFromSourceFile(QOpenGLShader::Fragment, "resources/shader.fs");
m_program->link();
```

- Shadery wczytujemy z osobnych plików i kompilujemy za pomocą funkcji `addShaderFromSourceFile()`
- Różne typy shaderów łączymy w jeden program poleceniem `link()`

- Dostępne typy shaderów w OpenGLu:
  - **Vertex Shader**: odpowiada za przetwarzanie wierzchołków i transformację do Clip Space
  - **Tessalation Shader**: pozwala dzielić kształty na mniejsze, tym samym je wygładzając
  - **Geometry Shader**: operuje na kształtach i pozwala na zmianę geometrii
  - **Fragment Shader**: ustawia kolor dla każdego fragmentu obrazu (piksela)
  - **Compute Shader**: przeznaczony do wykonywania obliczeń nie związanych bezpośrednio z renderingiem

- Na potrzeby tych zajęć będziemy analizować tylko shadery wierzchołków i fragmentów

Wczytywanie shaderów

Potok graficzny

Zmienne w shaderach

Atrybuty

Uniformy

Clip Space

Tworzenie kształtów

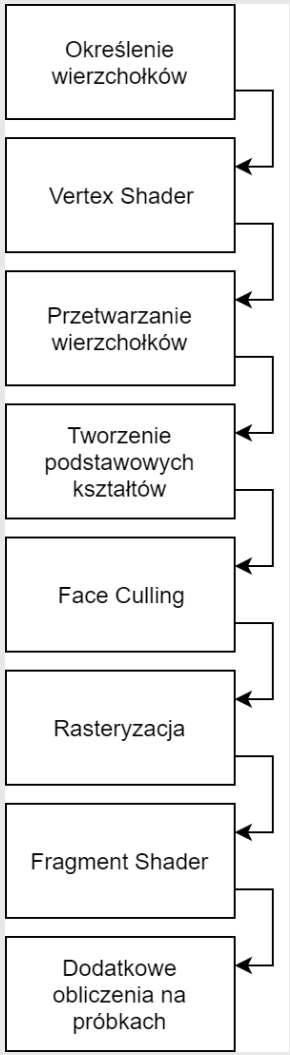
Face Culling

Rasteryzacja

Test głębokości

Efekt komiksowy

Uproszczony potok graficzny



→ Przesłanie wierzchołków do karty graficznej

→ Przekształcenie wierzchołków do Clip Space

→ Przekształcenie do Screen Space

→ Łączenie punktów w linie, trójkąty, czworokąty itp.

→ Usuwanie niewidocznych ścian

→ Wypełnianie wielokątów i zamiana kształtów na piksele

→ Ustawianie kolorów pikseli

→ Przeprowadzanie testów głębokości, mieszania kolorów itp.

Wczytywanie shaderów
Potok graficzny
Zmienne w shaderach
Atrybuty
Uniformy
Clip Space
Tworzenie kształtów
Face Culling
Rasteryzacja
Test głębokości
Efekt komiksowy

Typy zmiennych w shaderach

- `Attribute` – zmienna podawana na wejście shadera wierzchołków, unikalna dla każdego wierzchołka
- `Uniform` – zmienna, która jest ustawiana dla całego przebiegu shadera, więc jest jednakowa dla wszystkich jego wywołań
- `Varying` – zmienne tworzone w shaderze wierzchołków, które mogą być później używane we shaderze fragmentów
- Zmienne mogą mieć dodatkowo zdefiniowany rozmiar określający poziom precyzji: `highp` – 16 bitów, `mediump` – 10 bitów, `lowp` – 8 bitów
- W shaderze wierzchołków na końcu musi się znaleźć ustawienie wektora `gl_Position`, które określa położenie wierzchołka w Clip Space
- Shader fragmentów z kolei kończy się ustawieniem `gl_FragColor`, czyli kolorem fragmentu

Wczytywanie shaderów
Potok graficzny
Zmienne w shaderach
Atrybuty
Uniformy
Clip Space
Tworzenie kształtów
Face Culling
Rasteryzacja
Test głębokości
Efekt komiksowy

Ustawianie atrybutów

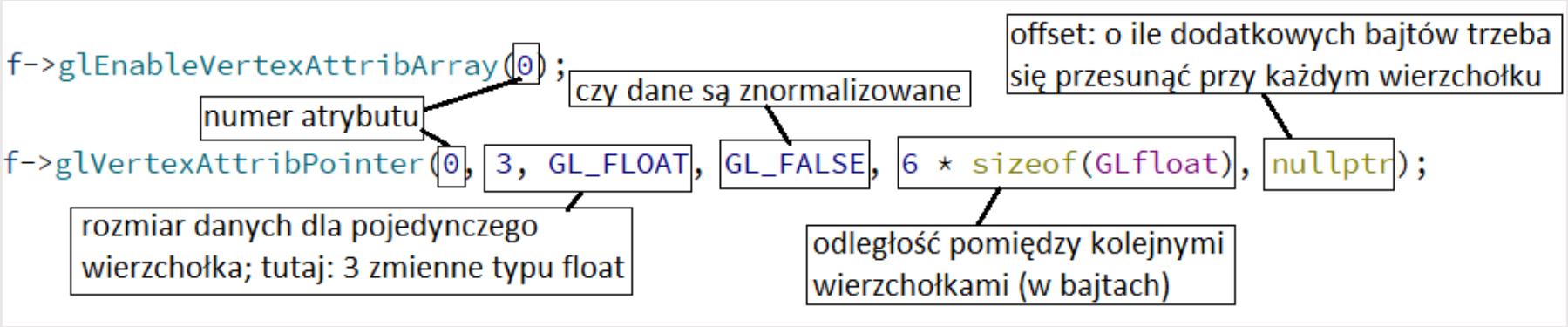
- Ustawienie atrybutów wymaga stworzenia specjalnego bufora, w którym przechowywane będą dane wszystkich wierzchołków:

```
m_vbo.create(); //stworzenie bufora
m_vbo.bind(); //podpięcie bufora, w celu przeprowadzenia na nim operacji
m_vbo.allocate(constData(), dataSize); //wstawienie danych do bufora
```

- Następnym krokiem jest określenie, w jaki sposób zapisaliśmy te dane – w tym celu tworzymy tablicę VAO (Vertex Array Object), która zapamięta wszystkie ustawienia (dzięki czemu wystarczy to zrobić tylko raz dla każdego obiektu):

```
m_vao.create(); // stworzenie obiektu
m_vao_binder = new QOpenGLVertexArrayObject::Binder(&m_vao); // podpięcie obiektu
```

- Na koniec definiujemy, co znajduje się w ustawionym przez nas buforze:



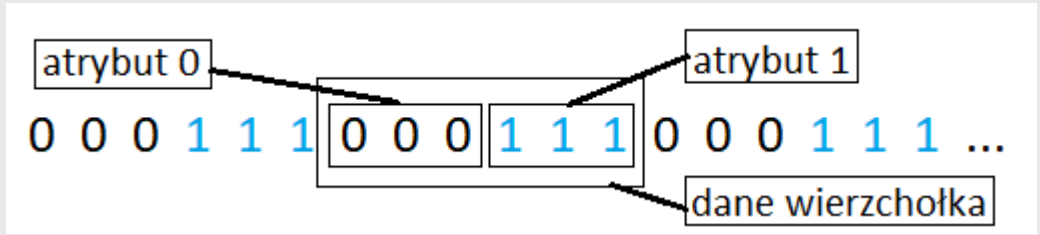
Wczytywanie shaderów
Potok graficzny
Zmienne w shaderach
Atrybuty
Uniformy
Clip Space
Tworzenie kształtów
Face Culling
Rasteryzacja
Test głębokości
Efekt komiksowy

Atrybuty – c.d.

Przykład:

```
f->glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat), nullptr);
f->glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat), reinterpret_cast<void *>(3 * sizeof(GLfloat)));
```

Dane będą ułożone w ten sposób:



Aby w konkretnym shaderze użyć takiego zestawu atrybutów, trzeba tylko nadać nazwy numerom:

```
m_program->bindAttributeLocation("vertex", 0);
m_program->bindAttributeLocation("normal", 1);
```

Później można ich użyć w shaderze w ten sposób:

```
attribute vec3 vertex;
attribute vec3 normal;
```

Wczytywanie shaderów
Potok graficzny
Zmienne w shaderach
Atrybuty
Uniformy
Clip Space
Tworzenie kształtów
Face Culling
Rasteryzacja
Test głębokości
Efekt komiksowy

Ustawianie uniformów

- Ustawienie uniformów jest znacznie prostsze, ponieważ potrzebujemy tylko jednego zestawu danych dla wszystkich wierzchołków

```
uniform highp vec3 modelColor;
```

- Na początku należy pobrać lokalizację uniformu w shaderze:

```
m_program->bind(); //podpięcie programu, w którym szukamy zmiennej
m_modelColorLoc = m_program->uniformLocation("modelColor");
```

- Następnie można ustawić wartość dla danego uniformu:

```
m_program->setUniformValue(m_modelColorLoc, QVector3D(0.3f, 0.5f, 0.0f));
```

Wczytywanie shaderów
Potok graficzny
Zmienne w shaderach
Atrybuty
Uniformy
Clip Space
Tworzenie kształtów
Face Culling
Rasteryzacja
Test głębokości
Efekt komiksowy

Konwersja do Clip Space

- Wartości uzyskiwane w shaderze wierzchołków muszą być zapisane do Clip Space
- Konwersja do Clip Space odbywa się poprzez przemnożenie lokalnej pozycji wierzchołków przez macierze modelu, widoku i projekcji
- Wszystkie wierzchołki znajdujące się poza Clip Space (czyli o współrzędnych mniejszych od -1.0 lub większych od 1.0) są odrzucane
- Kolejnym etapem jest konwersja do Screen Space: współrzędne -1 do 1 są zamieniane na współrzędne ekranu (np. 0 do 1920). Rozmiar tego obszaru określamy za pomocą polecenia:

```
glViewport(0, 0, 1920, 1080);
```

Wczytywanie shaderów
Potok graficzny
Zmienne w shaderach
Atrybuty
Uniformy
Clip Space
Tworzenie kształtów
Face Culling
Rasteryzacja
Test głębokości
Efekt komiksowy

Łączenie punktów w kształty

- Punkty uzyskane w poprzednich krokach są teraz łączone w kształty
- To, jakie kształty uzyskujemy, zależy od wywołanego przez nas polecenia do rysowania:

```
glWidget->glDrawArrays(GL_TRIANGLES, 0, 15);
```

- W ten sposób wywołaliśmy rysowanie dla 15 wierzchołków, które zostaną połączone w trójkąty (GL_TRIANGLES): każde kolejne 3 wierzchołki to jeden trójkąt

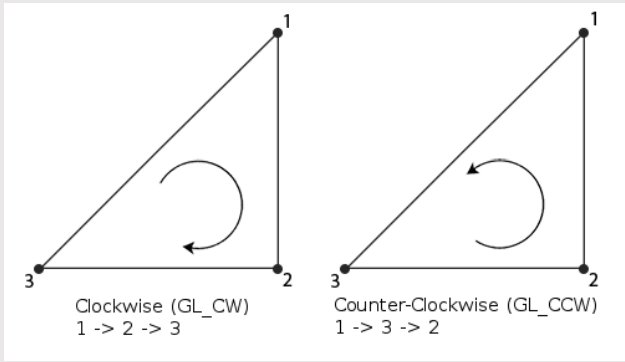
Wczytywanie shaderów
Potok graficzny
Zmienne w shaderach
Atrybuty
Uniformy
Clip Space
Tworzenie kształtów
Face Culling
Rasteryzacja
Test głębokości
Efekt komiksowy

Face Culling

- W kolejnym etapie, z wszystkich kształtów uzyskanych wcześniej, odrzucane są ściany przesłanianie
- Każdy trójkąt ma dwie strony, z których zawsze widoczna może być tylko jedna
- W celu zdefiniowania, która ściana jest która, używamy jednego z poleceń:

```
glFrontFace(GL_CCW);
```

```
glFrontFace(GL_CW);
```



- Do ukrycia ścian tylnych wykorzystujemy polecenie:

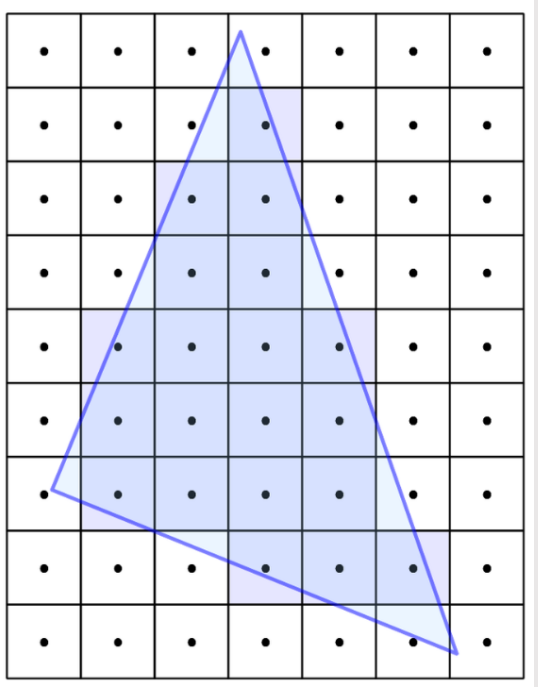
```
glCullFace(GL_BACK);
```

Rasteryzacja i shader fragmentów

Wczytywanie shaderów
Potok graficzny
Zmienne w shaderach
Atrybuty
Uniformy
Clip Space
Tworzenie kształtów
Face Culling
Rasteryzacja
Test głębokości
Efekt komiksowy

- W tym etapie wszystkie kształty, które wcześniej uzyskaliśmy są zamieniane na konkretne piksele
- Każdy z zaznaczonych pikseli został wyliczony jako znajdujący się w obrębie trójkąta, więc zostaje dla niego wykonany shader fragmentów
- Fakt, że nie można dokładnie odwzorować linii na siatce pikseli powoduje powstanie efektu „schodków”, tzw. aliasing
- W celu redukcji tego efektu, można zastosować antialiasing, który dla sąsiednich pikseli wylicza kolory pośrednie pomiędzy obiektem a tłem, wygładzając krawędzie
- Aby uzyskać taki efekt, należy w kostruktorze klasy GLWidget umieścić kod:

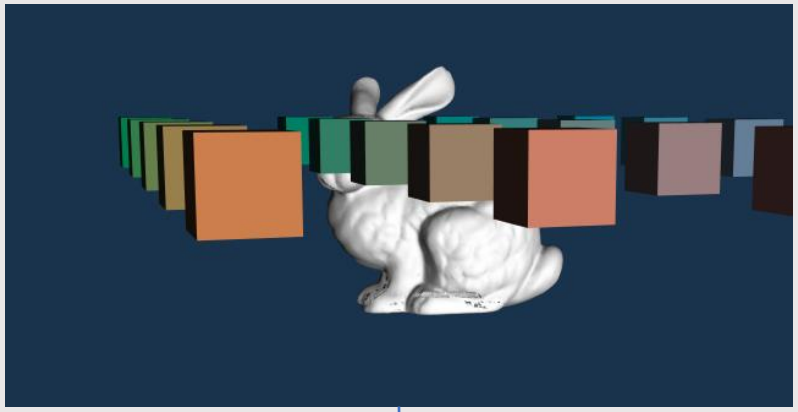
```
QSurfaceFormat format;
format.setSamples(16);
setFormat(format);
```



Wczytywanie shaderów
Potok graficzny
Zmienne w shaderach
Atrybuty
Uniformy
Clip Space
Tworzenie kształtów
Face Culling
Rasteryzacja
Test głębokości
Efekt komiksowy

Dodatkowe obliczenia

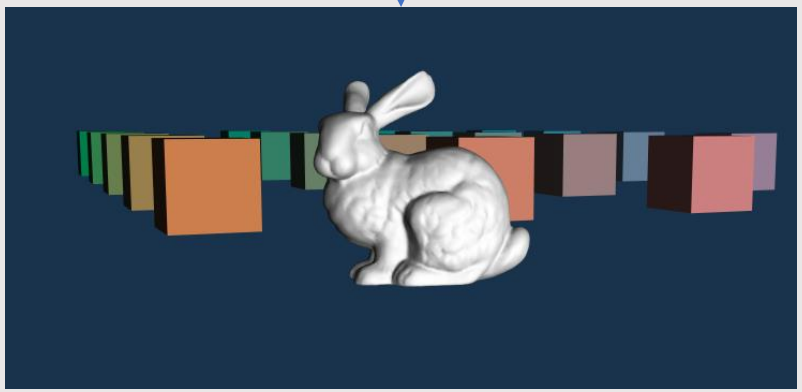
- Rysując kolejne obiekty wielokrotnie dojdzie do sytuacji, w której konkretny piksel jest wielokrotnie nadpisywany przez kolejne kształty
- Może to prowadzić do takiej sytuacji:



- Problem ten można rozwiązać tworząc bufor głębokości, który zapisuje dla każdego piksela wartość odległości:

```
glEnable(GL_DEPTH_TEST);
```

- Jeśli rysowany fragment znajduje się dalej niż już zapisana wartość, to jest on ignorowany, dzięki czemu głębokość działa poprawnie:



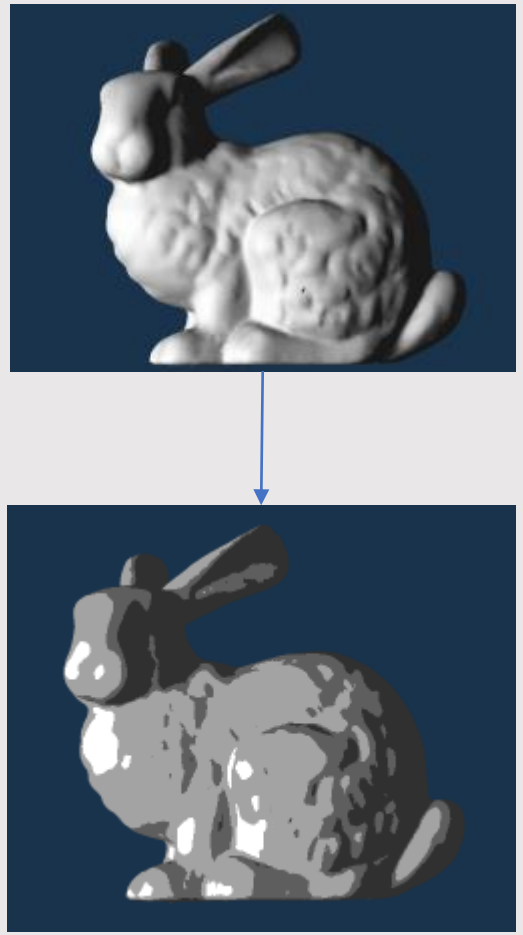
Wczytywanie shaderów
Potok graficzny
Zmienne w shaderach
Atrybuty
Uniformy
Clip Space
Tworzenie kształtów
Face Culling
Rasteryzacja
Test głębokości
Efekt komiksowy

Efekt komiksowy - progowanie

- Aby uzyskać renderowanie komiksowe (toon shading), musimy wykonać dwa etapy: uwidocznienie konturu obiektu oraz progowanie koloru

- Aby uzyskać progowanie koloru, należy zmienić parametr odpowiadający za gładkie cieniowanie na wartości dyskretne, np. w taki sposób:

```
if (cosNL > 0.95)
    cosNL = 1.0f;
else if (cosNL > 0.5)
    cosNL = 0.6f;
else if (cosNL > 0.25)
    cosNL = 0.3f;
else
    cosNL = 0.1f;
```



Wczytywanie shaderów
Potok graficzny
Zmienne w shaderach
Atrybuty
Uniformy
Clip Space
Tworzenie kształtów
Face Culling
Rasteryzacja
Test głębokości
Efekt komiksowy

Efekt komiksowy - kontur

- Stworzenie konturu wymaga wyrenderowania osobno konturu i obrazu i połączenia ich ze sobą
- Najpierw zmieniamy sposób renderowania. Zamiast renderować wypełnione trójkąty, rysujemy tylko krawędzie:

```
glPolygonMode(GL_BACK, GL_LINE);
```

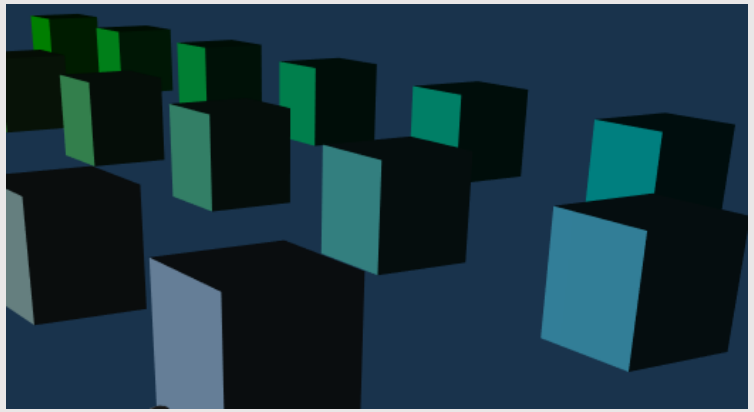
- Następnie zmieniamy culling ze ścian tylnych na ściany przednie:

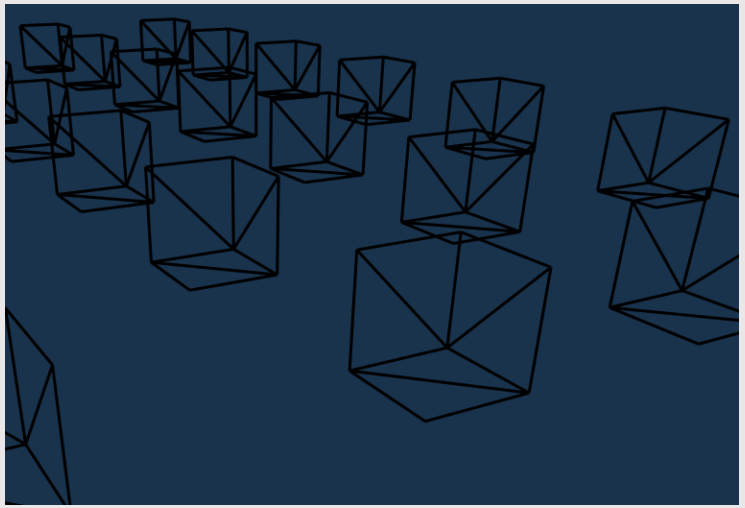
```
glCullFace(GL_FRONT);
```

- Dzięki temu widzimy tylko kontur tylnych ścian, a nie wszystkie krawędzie widocznych trójkątów
- Zmieniamy też grubość linii:

```
glLineWidth(3.0f);
```

- Trzeba również ustawić kolor krawędzi na czarny





Wczytywanie shaderów
Potok graficzny
Zmienne w shaderach
Atrybuty
Uniformy
Clip Space
Tworzenie kształtów
Face Culling
Rasteryzacja
Test głębokości
Efekt komiksowy

Efekt komiksowy – kontur – c.d.

- Następnie cofamy wszystkie wykonane zmiany:

```
glPolygonMode(GL_BACK, GL_POLYGON);
glCullFace(GL_BACK);
```

- … i renderujemy scenę po raz kolejny, tym razem w całości

- W ten sposób łączymy dwa obrazy tworząc nowy, składający się z normalnego renderingu i konturów: