WSTĘP DO GLSL

Bartłomiej Filipek
www.ii.uj.edu.pl/~filipek

Plan

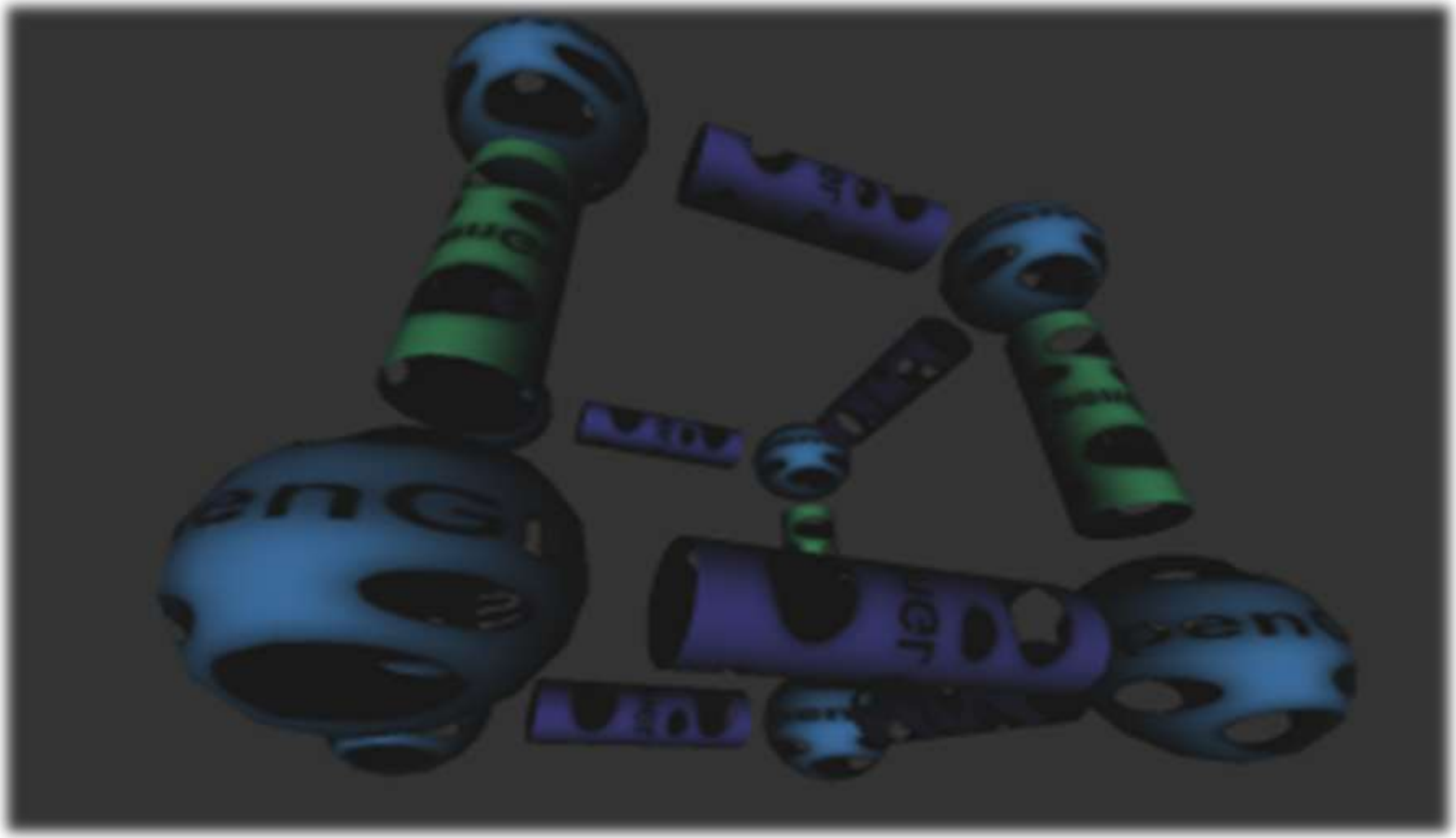
- Nasz Cel
- Prehistoria… krótki wstęp…
- Nowa era!
- Vertex Shaders
- Fragment Shaders
- Podstawy GLSL
- Obsługa GLSL z API OpenGL
- Dodajmy parę efektów!
- Podsumowanie
- Dodatkowe materiały

Źródła

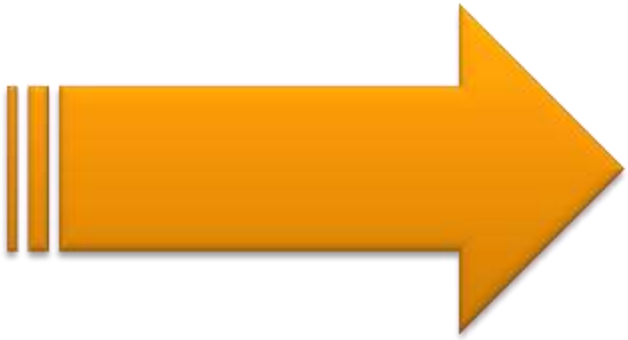
- OpenGL SuperBible, 4th edition,
- www.lighthouse.com
- www.opengl.org/sdk
- http://www.typhoonlabs.com/

- Prezentacja i przykłady są dostępna pod adresem: www.ii.uj.edu.pl/~filipek/tutorials

Nasz Cel



Nasz Cel

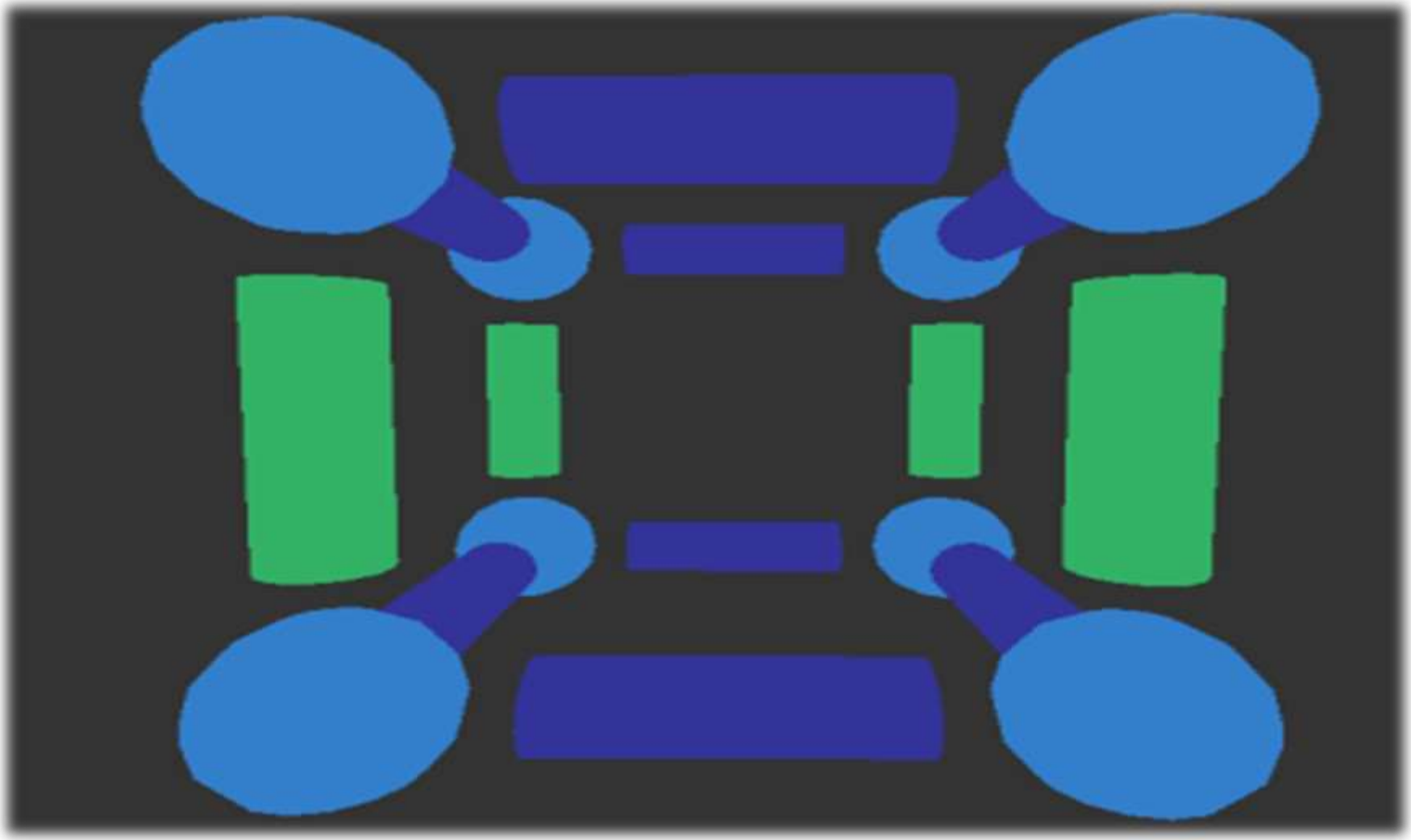


Projekt: Start

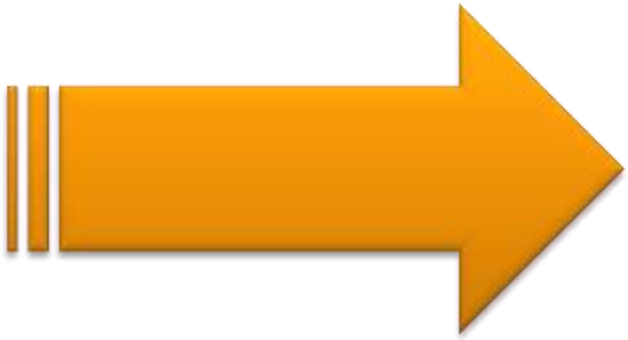
Nasz Cel

- Proste oświetlenie per-pixel
- Teksturowanie i mieszanie kolorów
- animacja wykonywana na shaderze (oparta na czasie)
- Maskowanie kolorów – tworzenie „dziur” w obiektach
- Inne dodatki ☺

A wystartujemy od:

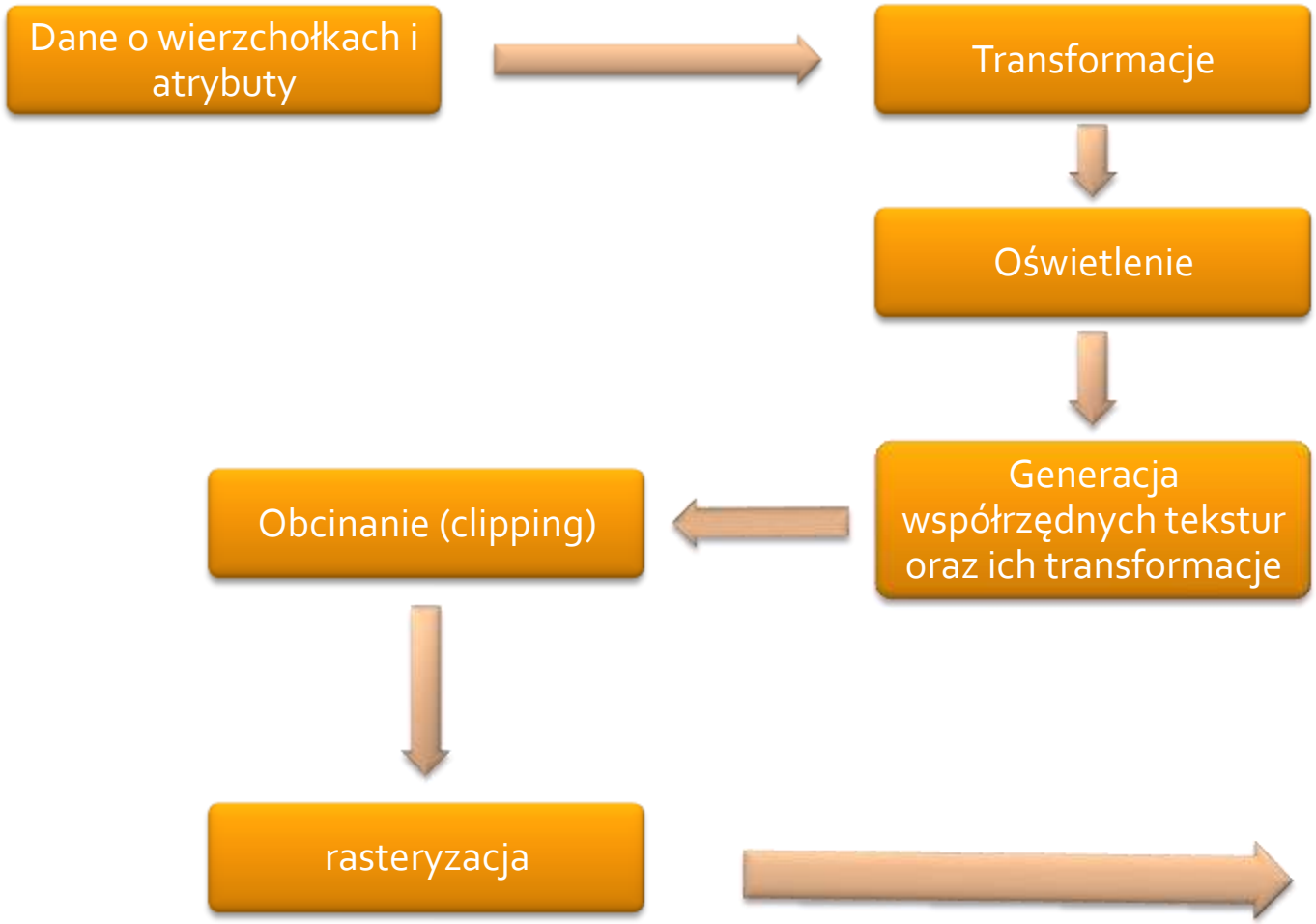


Początki…

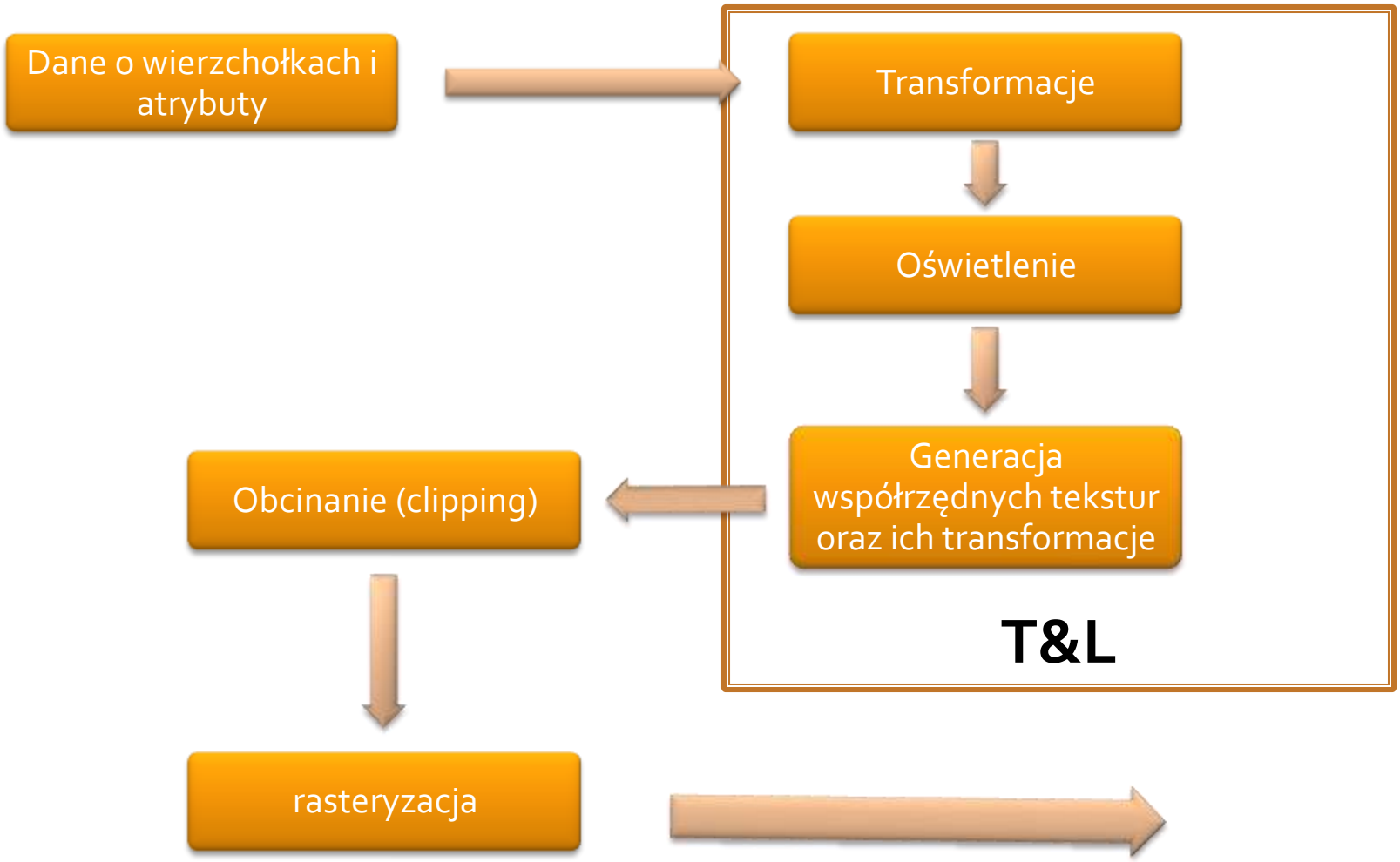


Projekt: Fixed

Prehistoria... krótki wstęp...

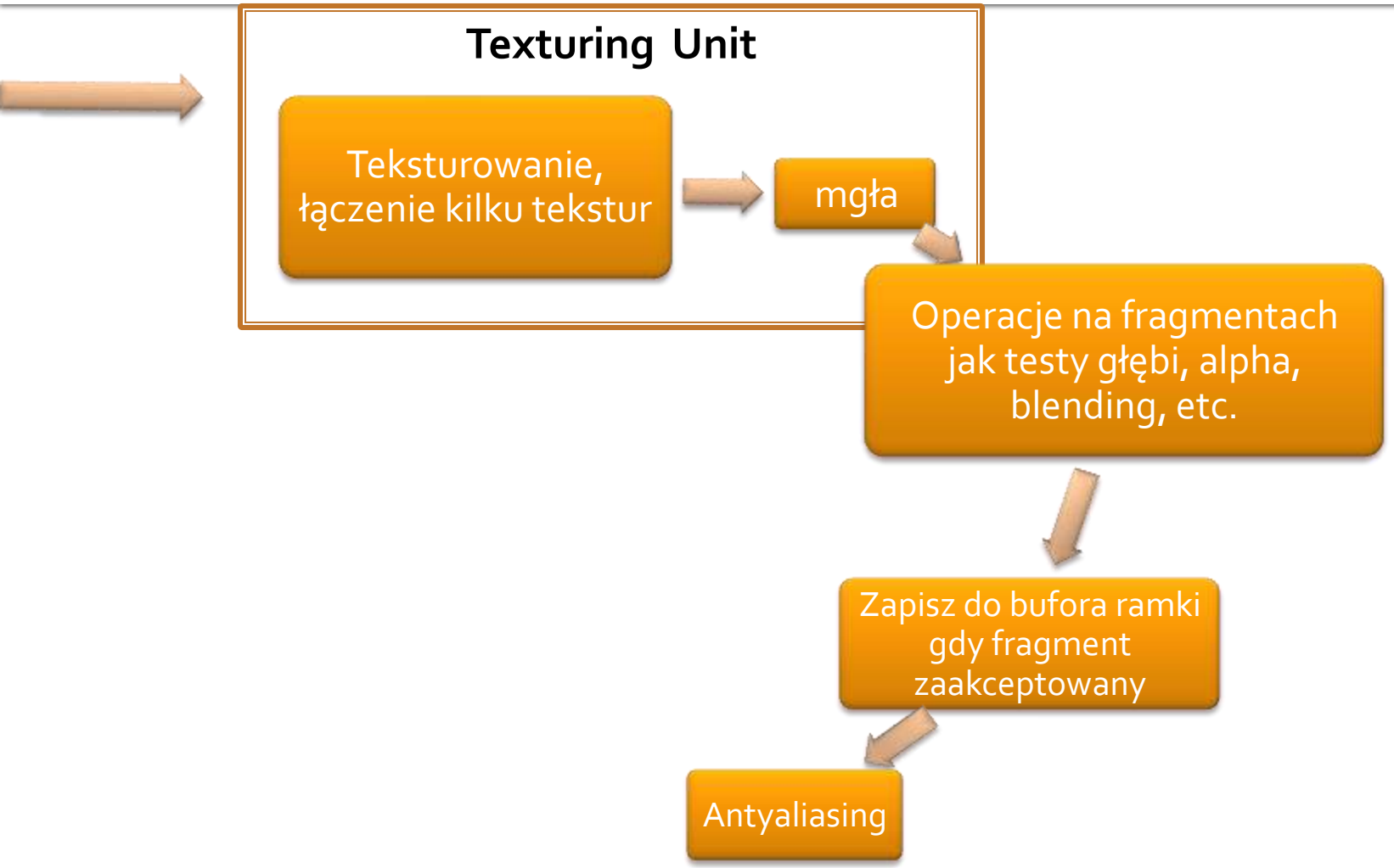


Prehistoria... krótki wstęp...



Prehistoria... krótki wstęp...

Prehistoria... krótki wstęp...



Starocie...

```
glEnable(GL_LIGHTING);

glActiveTexture(GL_TEXTURE0);
glTexEnvi( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE );

glDisable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, texFloor);

glActiveTexture(GL_TEXTURE1);
glTexEnvi( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_ADD );

glDisable(GL_TEXTURE_2D);
if (g_renderReflection) {
        glEnable(GL_TEXTURE_CUBE_MAP);
        glBindTexture(GL_TEXTURE_CUBE_MAP, reflectionTex);
        glEnable(GL_TEXTURE_GEN_S);
        glEnable(GL_TEXTURE_GEN_T);
        glEnable(GL_TEXTURE_GEN_R);

        glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP);
        glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP);
        glTexGeni(GL_R, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP);
```

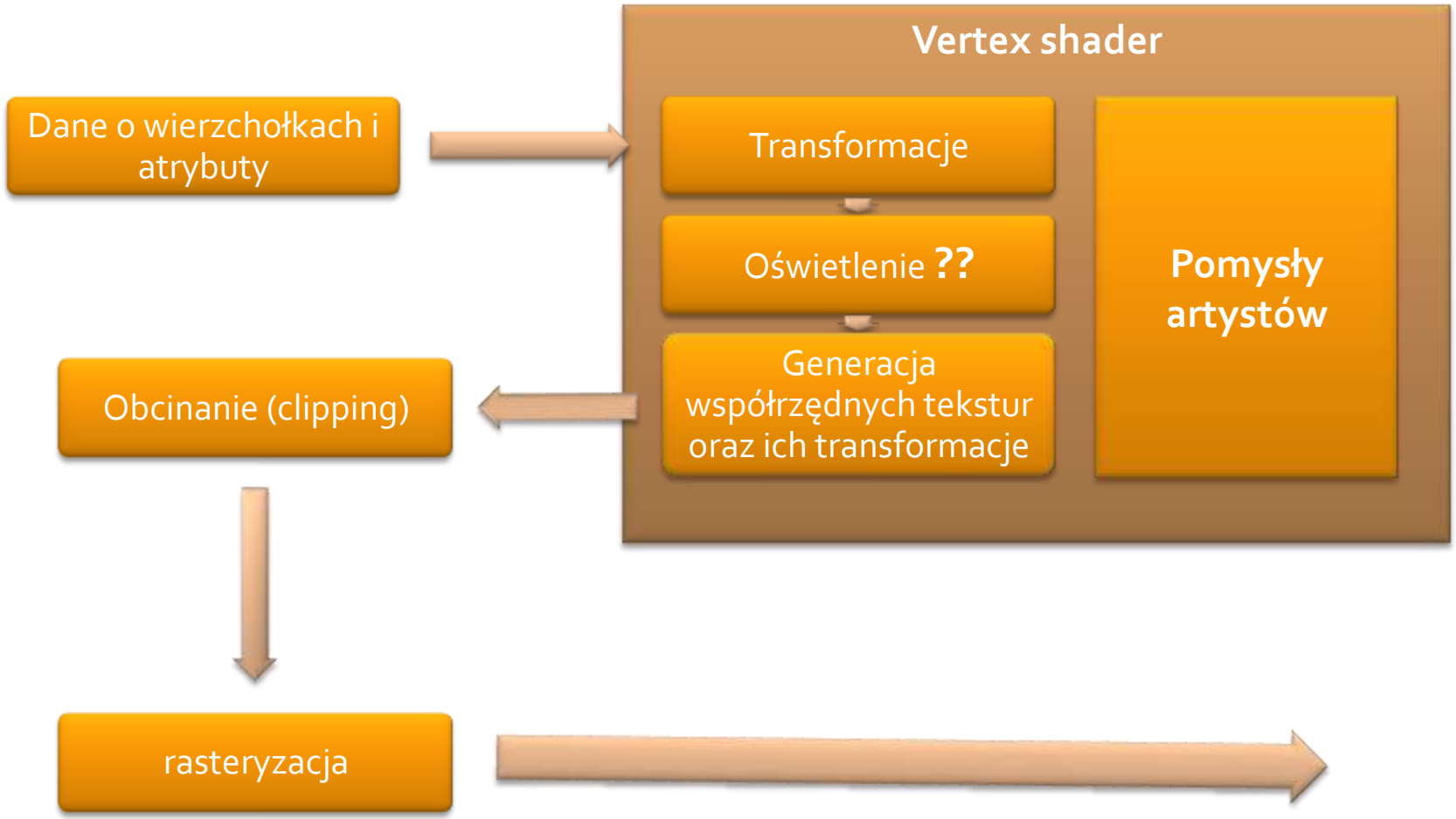
Dlaczego potrzebna była zmiana?

- Graficy i programiści czuli się ograniczeni przez statyczne i ustalone funkcje wewnętrzne OpenGL'a.
- Aby stworzyć efekty trzeba było się nieźle natrudzić przy ustawianiu kolejnych parametrów stanów wewnętrznych OpenGL'a. **„Tu coś wyłącz… tu coś ustaw, etc, etc…"**
- Brakowało elastyczności – **„dlaczego nie mogę stworzyć czegoś nowego?"**
- Moc obliczeniowa kart graficznych jest tak duża, że fixed pipeline był dla nich „wąskim gardłem".

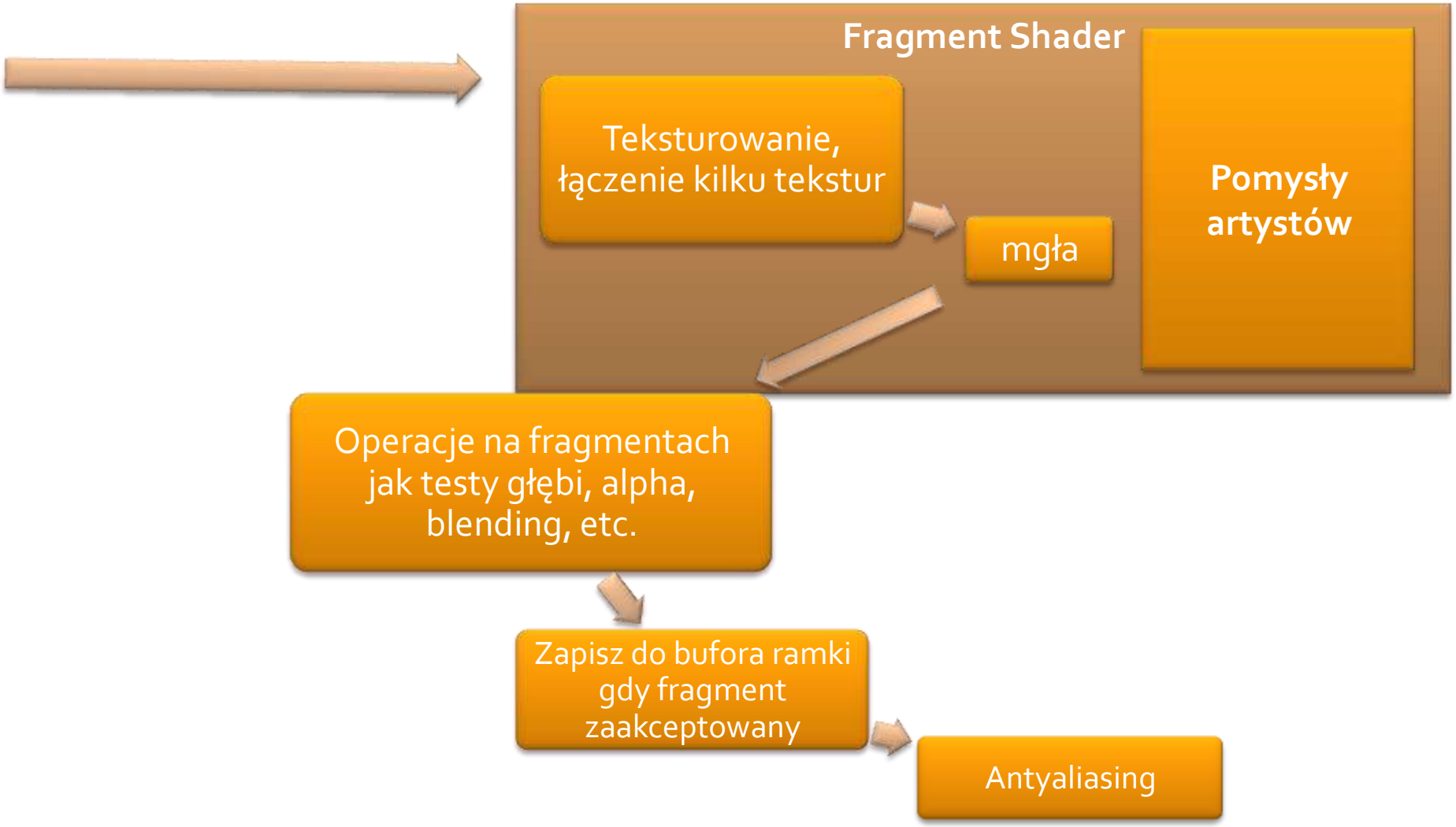
Nowa era!

- Powstały dwie jednostki: vertex i fragment shadery.
- Verte shader ma na celu wykonać wszelkie operacje na wierzchołku – transformacje, ustawianie kolorów, współrzędne tekstur, normalne…
- Fragment shader zastępuje głównie teksturowanie, pokrywanie materiałem obiekty.
- **Od teraz wszystko programujemy, a nie ustawiamy!**

Nowy potok renderowania



Nowy potok renderowania...



Vertex Shaders

Vertex Shaders

- Wykonywane dla każdego wierzchołka, wiec może być wiele shaderów wykonywanych równolegle, dla kilku wierzchołków naraz!
- Może:
  - Przekształcić wierzchołki z przestrzeni obiektu do przestrzeni obcinania (clip-space)
  - **Przekazać** kolor, normalne i inne atrybuty do pixel shadera.
  - Obliczyć współrzędne tekstur…
  - I wiele innych obliczeń.
- **Minimum to ustawić wartość dla gl_Position**
- Obliczone wartości (jak kolor, normale…) są interpolowane i przekazywane do fragment shader'a

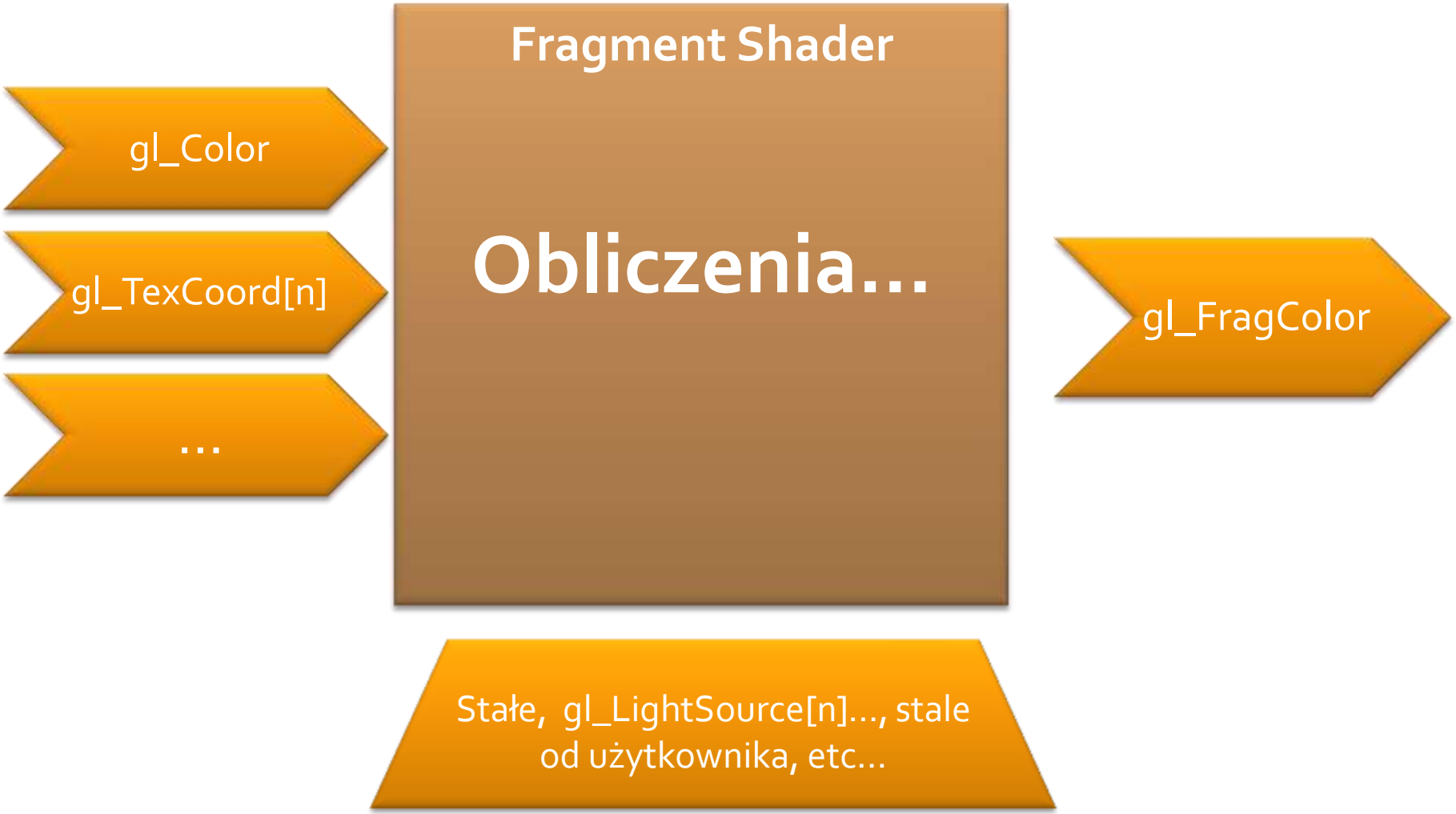
Najprostszy vertex shader

```
void main()
{
gl_Position = gl_ModelViewProjectionMatrix *
              gl_Vertex;

gl_FrontColor = gl_Color *2.0 -
                vec4(0.1, 0.1, 0.1, 0.0);

}
```

Gl_ModelViewProjectionMatrix – dostajemy od maszyny stanów OpenGL
gl_Vertex – dostajemy jako atrybut podawany w glVertex(…)
gl_Color – jako parametr z glColor(…)

Gl_Position – minimum, które trzeba wykonać w shaderze to właśnie ustawić tą wartość.
Gl_FrontColor – mówi dalszym etapom renderowania o kolorze dla przedniej ściany obiektu…

Fragment Shaders



Najprostszy Fragment Shader

```
void main()
{
 gl_FragColor = gl_Color;
 gl_FragColor.g = 1.0;
}
```

gl_FragColor – minimum które trzeba wykonać w Pixel shaderze – ustawić kolor fragmentu

Gl_Color – interpolowana wartość pochodząca od gl_FrontColor ustawionej w vertex shaderze

Jedna uwaga…

Gdy już zdecydujemy się na użycie jakiegoś shadera to w całości musimy zastąpić dany etap potoku renderowania.

Na przykład nie możemy tylko obliczyć pozycji wierzchołka i żądać od OpenGL'a aby sam sobie policzył dla tego samego wierzchołka oświetlenie.

Lub też obliczyć kolor w pixel shaderze i chcieć aby mgła policzyła się sama.

Podstawy GLSL

- Język bardzo podobny składniowo do C/C++, dlatego uczy się go bardzo szybko i sprawnie
- W związku z tym, że jest bardzo specjalizowany głównie skupiono się na łatwości obliczeń, mamy więc do dyspozycji multum funkcji matematycznych.
- na pewno nie ma umiejętności otwierania i zapisywania do pliku, czy wyrzucania wyjścia na konsole
- Gotowe klasy/struktury dla wektorów (vec2, vec3, vec4…) i macierzy (mat2, mat3…) wraz z wszelkimi operacjami na nich.
- Wbudowana obsługa czytania z tekstur za pomocą tzw. sampler'a
- Instrukcje warunkowe i pętle, instrukcje break, continue, return…
- Można pisać funkcje, tworzyć struktury.
- Specjalna instrukcja „discard" dla fragmentów, powoduje odrzucenie fragmentu i nie zapisanie go do bufora.
- Oczywiście można się komunikować s shaderami i ustawiać im odpowiednie wartości z głównego naszego programu.

GLSL – atrybuty zmiennych

- Uniform – można ją ustawiać jedynie pomiędzy wywołaniami glBegin/glEnd… innymi słowy: „stała dla konkretnego obiektu”.
- Varying – gdy ustawimy ją w vertex shaderze to fragment shader otrzyma tą wartość, ale odpowiednio interpolowaną. Np. normal, position, etc…
- Attrib (dla wierzchołków) – atrybut dla pojedynczego wierzchołka jak gl_Color, gl_Normal
- Const – stała na potrzeby wewnętrzne shadera
- Bez kwalifikatora – zmienna zwykła – globalna lub lokalna.

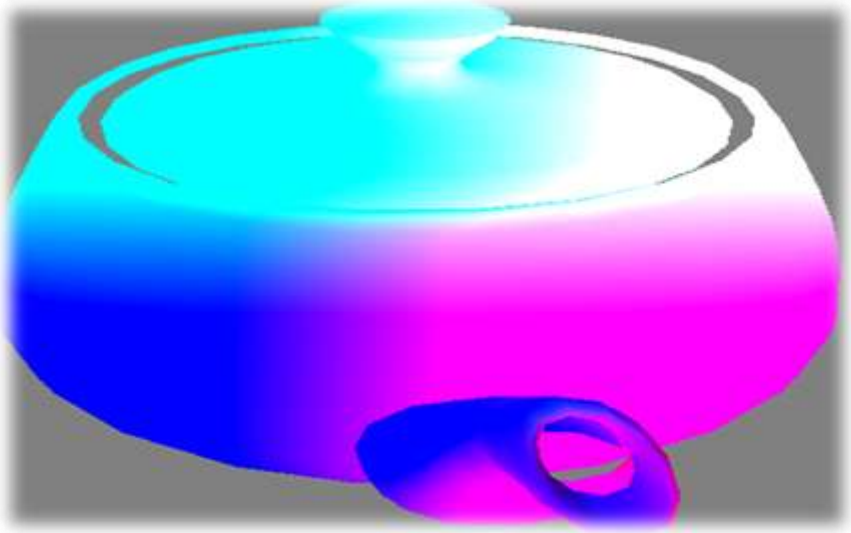
Przykłady shaderów – „kolor”

VERTEX Shader

```
varying vec4 position;
varying vec4 normal;

void main()
{
        normal = vec4(gl_Normal, 1.0);

        position =
        gl_ModelViewProjectionMatrix *
        gl_Vertex;

        gl_Position = position;
        gl_FrontColor = gl_Color;
}
```

PIXEL Shader

```
varying vec4 position;
varying vec4 normal;

void main()
{
    gl_FragColor = normal+position;
}
```



Shadery napisane w:
OpenGL Shader Designer 1.5
www.typhoonlabs.com

Przykłady shaderów – Diffuse light

VERTEX Shader

```
varying vec3 position;
varying vec3 normal;

void main()
{
    normal = gl_Normal;

    vec4 v = gl_ModelViewMatrix * gl_Vertex;

    position = v.xyz;

    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;

    gl_FrontColor = gl_Color;
}
```

Przykłady shaderów – Diffuse Light

PIXEL Shader

```
varying vec3 position;
varying vec3 normal;

void main()
{
    vec3 light = gl_LightSource[0].position.xyz - position;
    light = normalize(light);

    vec3 norm = normalize(normal);

    float lightDiffuse = max(0.0,dot(norm, light));

    gl_FragColor.rgb = lightDiffuse;
    gl_FragColor.a = 1.0
}
```

Shadery napisane w:
OpenGL Shader Designer 1.5
www.typhoonlabs.com

Podstawy GLSL

- GLSL Quick Ref – ściągawka ze składni i używania.
- TyphoonLabs – tu już trochę więcej na temat języka, bardzo polecam.

(Nie)typowe zastosowania

- Postprocessing (blur, motion blur, glow, etc..)
- Przeglądarka Fraktali
- Obliczenia współbieżne
- NVidia CUDA
- Fizyka
- Muzyka – generowanie i obróbka dźwięku

Obsługa GLSL z API OpenGL

```
void InitShaders() {
   GLint vertex_shader = glCreateShader(GL_VERTEX_SHADER);
   GLint fragment_shader=  glCreateShader(GL_FRAGMENT_SHADER);

   char *vs = ReadAllTextFromFile("shaders/basic.vert");
   char *fs = ReadAllTextFromFile("shaders/basic.frag");

   const char * vv = vs;
   const char * ff = fs;

   glShaderSource(vertex_shader, 1, &vv,NULL);
   glShaderSource(fragment_shader, 1, &ff,NULL);

   glCompileShader(vertex_shader);
   glCompileShader(fragment_shader);
   ... ...
   }
```

Obsługa GLSL z API OpenGL

```
Glint shader_program = glCreateProgram();
glAttachShader(shader_program, vertex_shader);
glAttachShader(shader_program, fragment_shader);

glLinkProgram(shader_program);

GLint timeElapsed = glGetUniformLocation(shader_program,
                                         "time");
Glint texture = glGetUniformLocation(shader_program,
                                     "texture");

glBindTexture(GL_TEXTURE_2D, gTexSample);
glUniform1i(texture, 0);
```

Obługa GLSL z API OpenGL

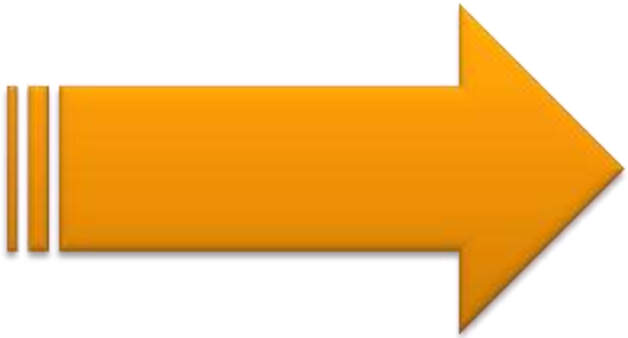
```
glUseProgram(shader_program);

glUniform1f(timeElapsed,
  (float)gTimer->GetTime());

… glBegin/glEnd …

glUseProgram(0);
```

Obługa GLSL z API OpenGL

```
glUseProgram(0);
glDeleteProgram(shader_program);
```

Obługa GLSL z API OpenGL

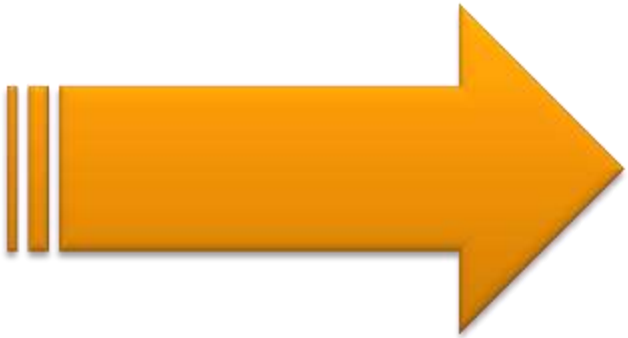
- Jak to wygląda w kodzie?



Projekt: Basic_shaders

Dodajmy parę efektów!

- Co można ciekawego zrobić?



Projekt: Start

Podsumowanie

- Pokazaliśmy „starocie” czyli jak działa mniej więcej statyczny potok renderowania i jakie niesie on ze sobą ograniczenia.
- Omówiliśmy jednostki vertex i pixel shader.
- Jak to może być użyte w OpenGL
- I zrobiliśmy ciekawy projekt.

- I mam nadzieje, że każdy zrozumiał, że shadery to jest naprawdę potęga.

Pytania?

Dodatkowe materiały

- Wszystko to co napisałem w slajdzie pt. „źródła” tej prezentacji.
- Shader Designer (TyphoonLabs) – można w tym pisać shadery i jest Open Source
- GPU Gems 1 – książka od Nvidii, udostępniona za darmo na ich stronach.
- CG i HLSL – inne wysoko poziomowe języki cieniowania.
- Kolorowanie Składni GLSL dla Visual Studio

Dziękuje za uwagę

www.ii.uj.edu.pl/~filipek