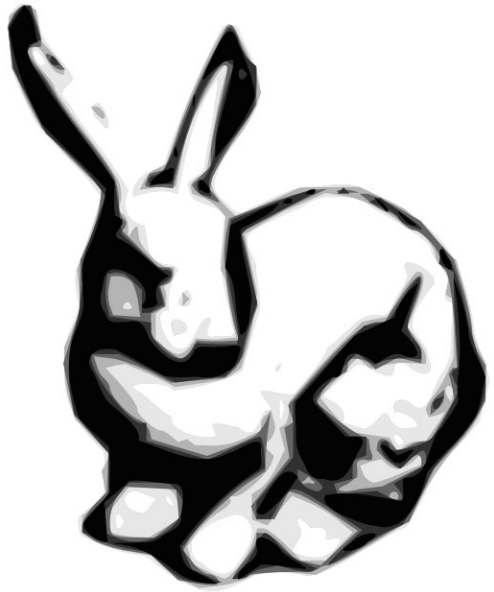
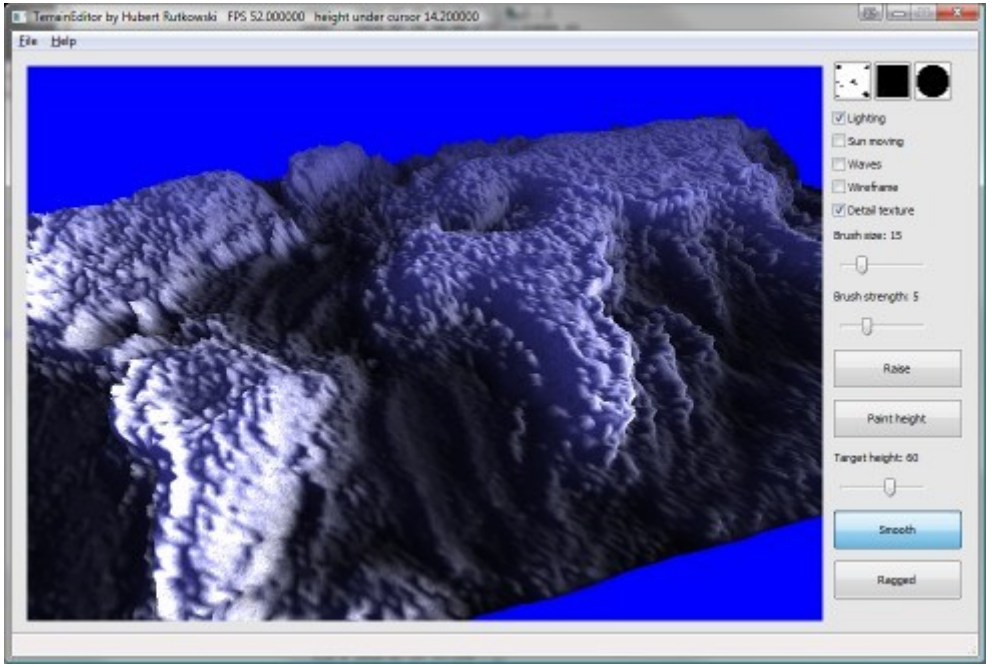
Potok graficzny i shadery

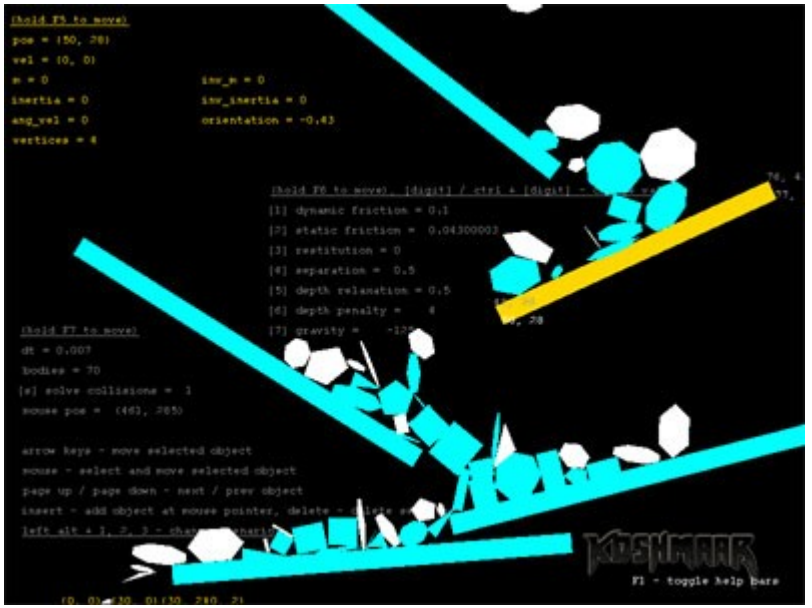


Hubert Rutkowski

1. Coś o mnie

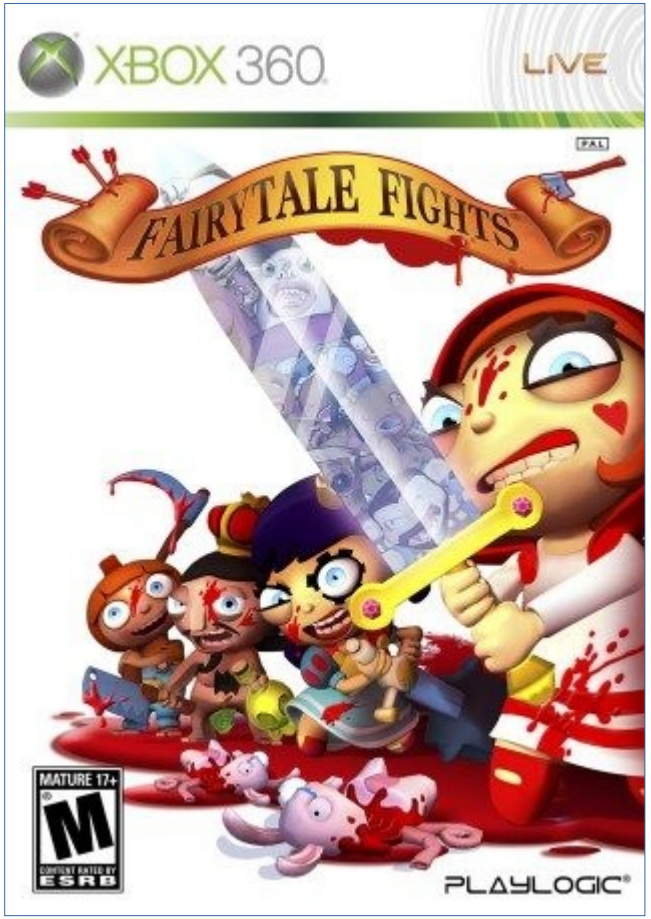

TerrainEditor by Hubert Rutkowski FPS 52.000000 height under cursor 14.200000
File Help
Lighting
Sun moving
Waves
Wireframe
Detail texture
Brush size: 15
Brush strength: 5
Raise
Paint height
Target height: 60
Smooth
Ragged


pencil


[hold F5 to move]
pos = (50, 20)
vel = (0, 0)
m = 0
inv_m = 0
inertia = 0
inv_inertia = 0
ang_vel = 0
orientation = -0.43
vertices = 4
[hold F6 to move]
[1] dynamic friction = 0.1
[2] static friction = 0.04300003
[3] restitution = 0
[4] separation = 0.5
[5] depth relaxation = 0.5
[6] depth penalty = 4
[hold F7 to move]
dt = 0.007
bodies = 70
mouse pos = (461, 265)
arrow keys - move selected object
mouse - select and move selected object
page up / page down - next / prev object
insert - add object at mouse pointer, delete - delete object
F1 - toggle help here


100
animals:
Caves of Sadness


Sadness
ALL RIGHTS RESERVED Nibris 2006
nibris


XBOX 360
LIVE
PAL
FAIRYTALE FIGHTS
MATURE 17+
M
ESRB
PLAYLOGIC


IGN.COM


Connection status: Server!
Disconnect

GRACE 2
ENGINE

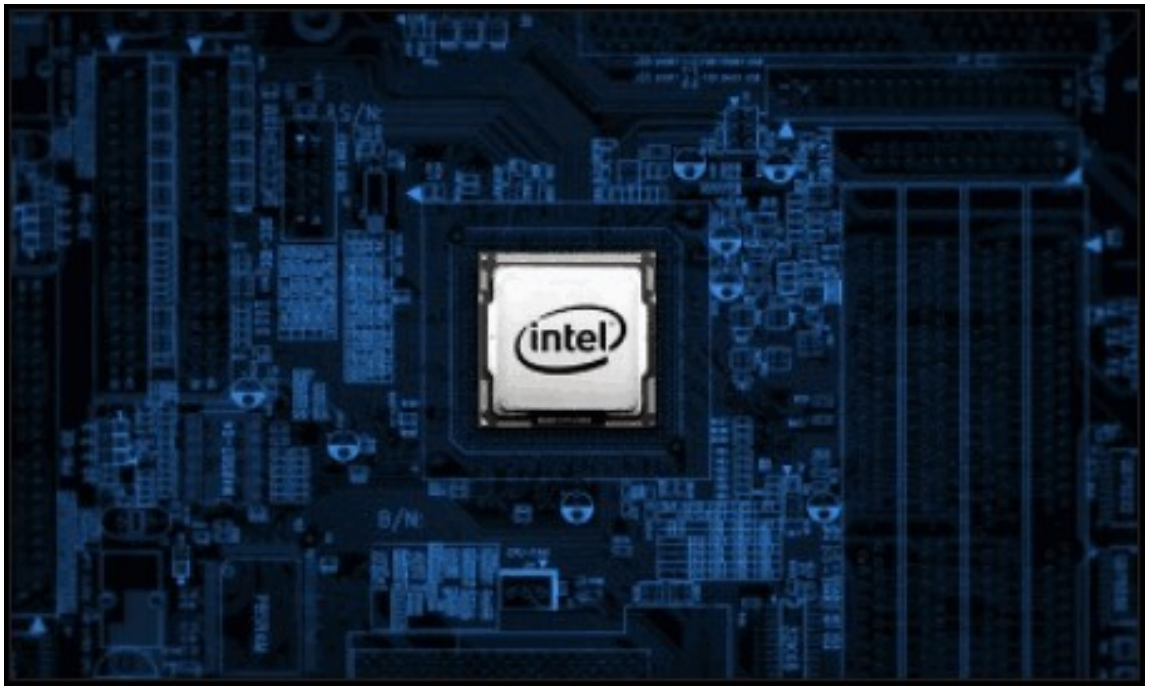

Ninja Cat
AND
Zombie Dinosaurs


Ninja Cat
foster the people
pal
uplink
up_
uplink
220

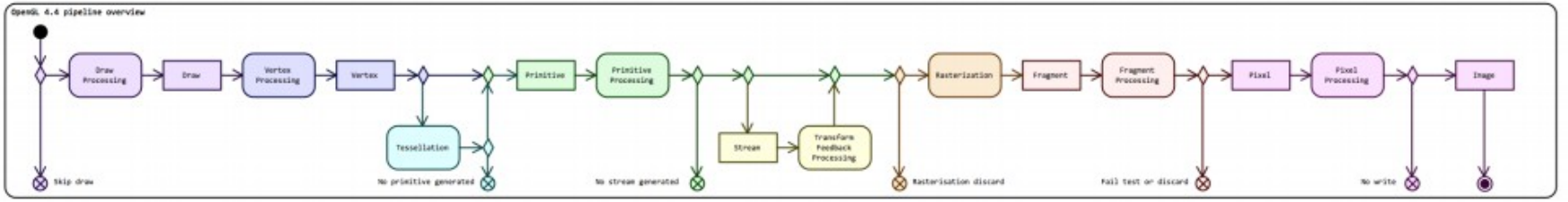

Ninja Cat
Raargh, Tyrannosaurus Rex
no fate, but what we make for ourselves
2,490

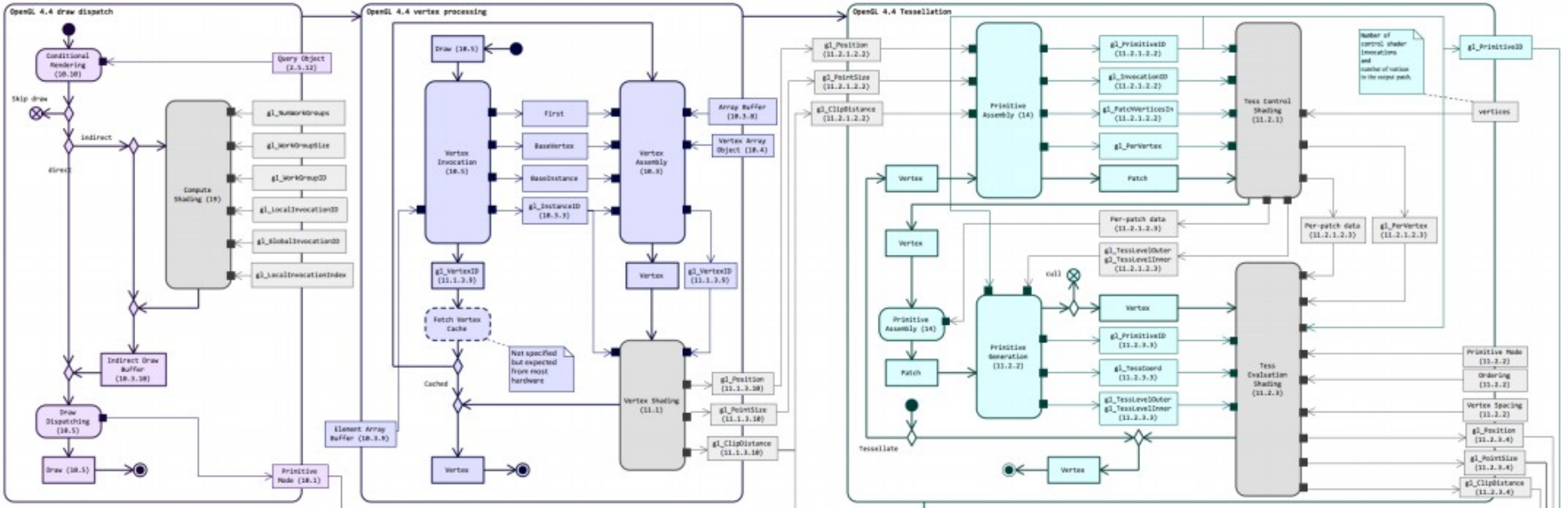
Zakład Technologii Gier

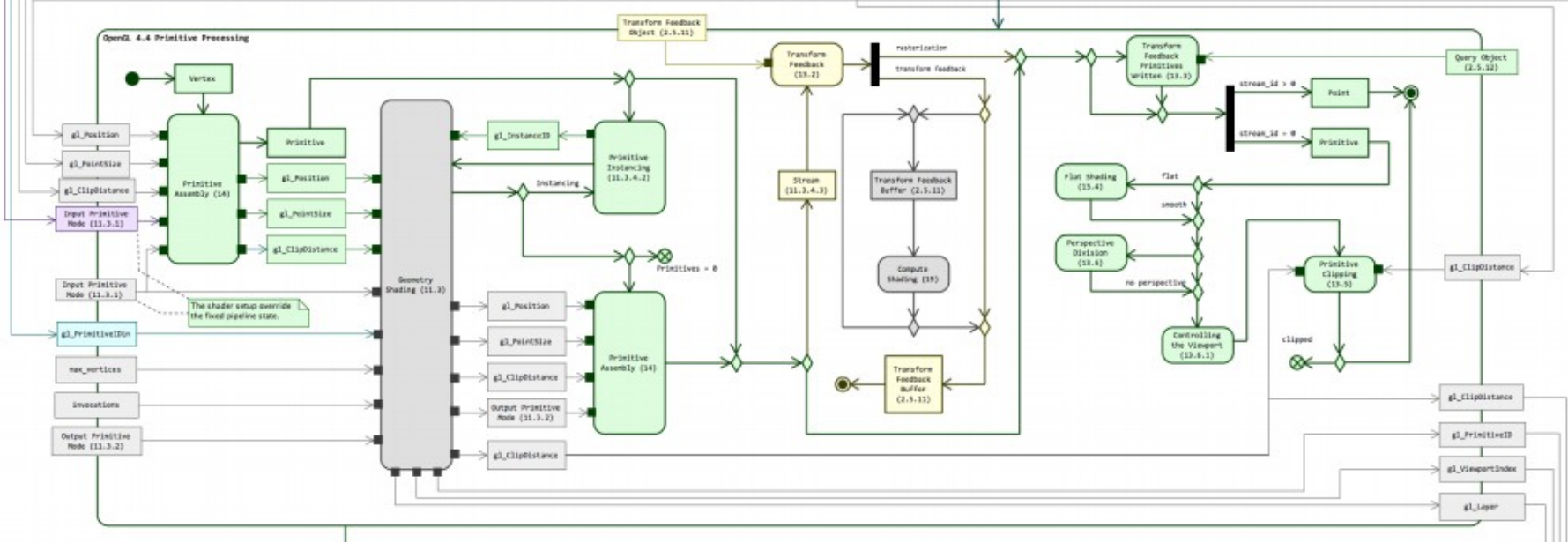
Uniwerytetu Jagiellońskiego


intel

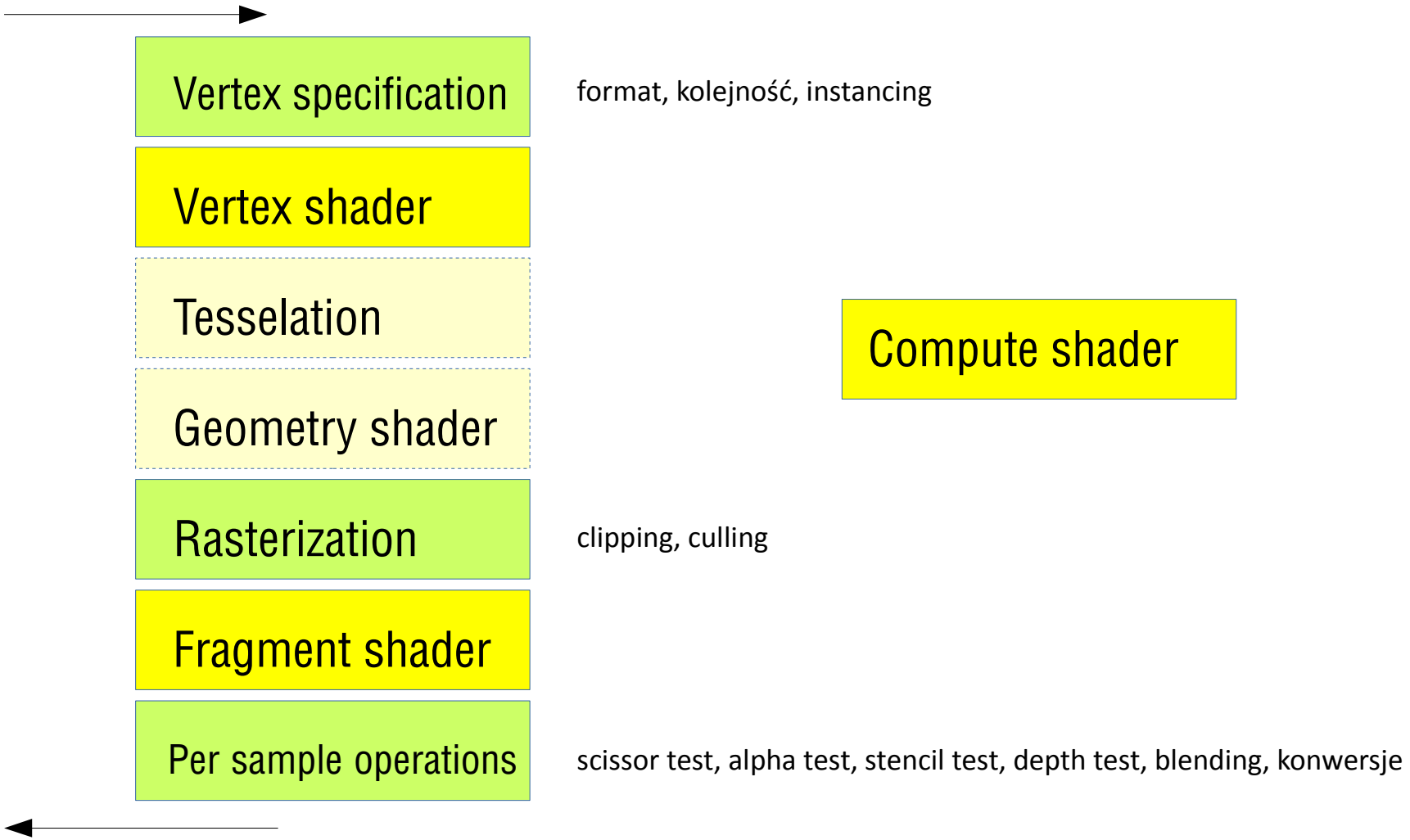
2. Potok graficzny
jak w OpenGL

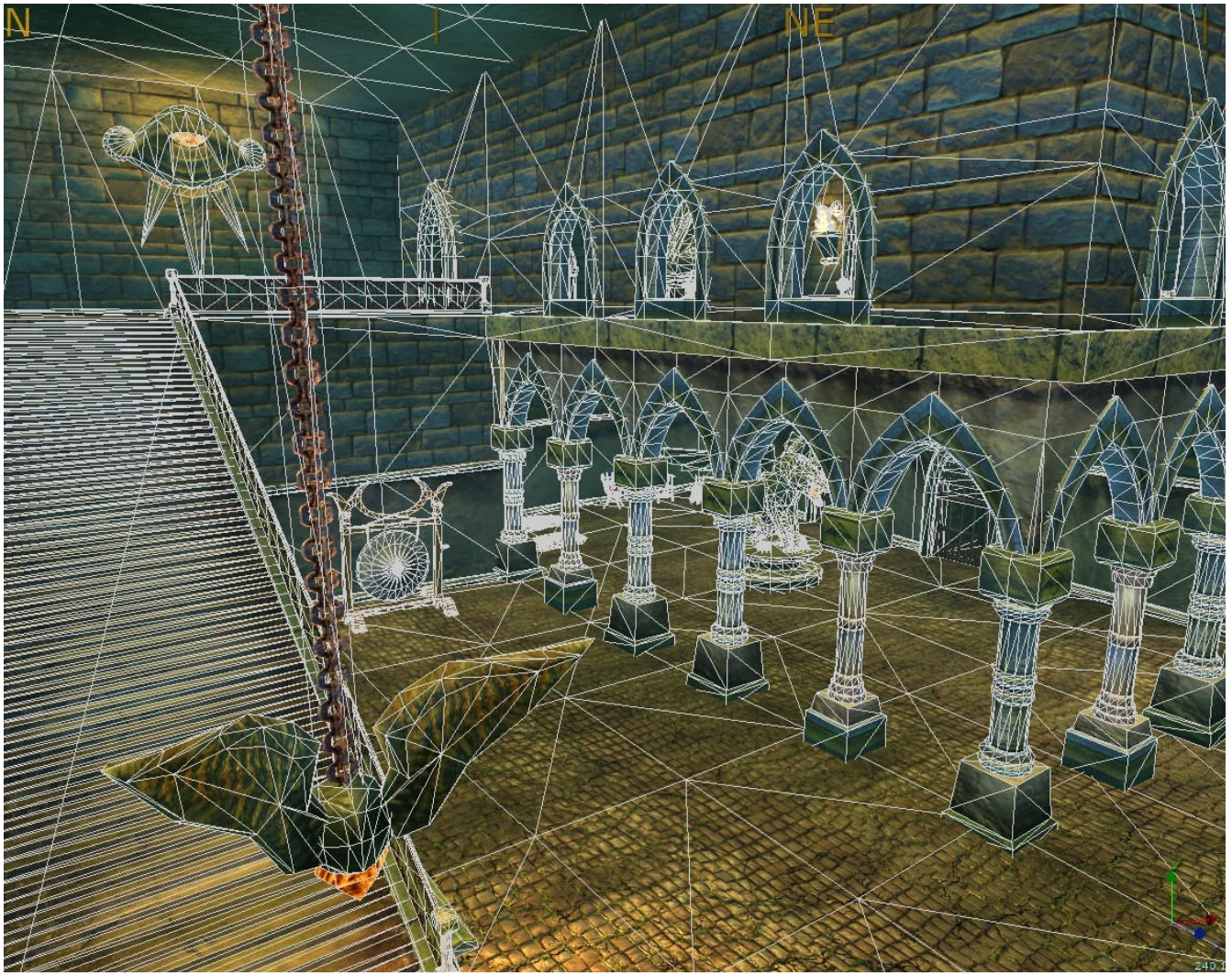

OpenGL 4.4 pipeline overview
Draw Processing
Draw
Vertex Processing
Vertex
Tessellation
Primitive
Primitive Processing
Stream
Transform Feedback Processing
Rasterization
Fragment
Fragment Processing
Pixel
Pixel Processing
Image
Skip draw
No primitive generated
No stream generated
Rasterization discard
Fail test or discard
No write

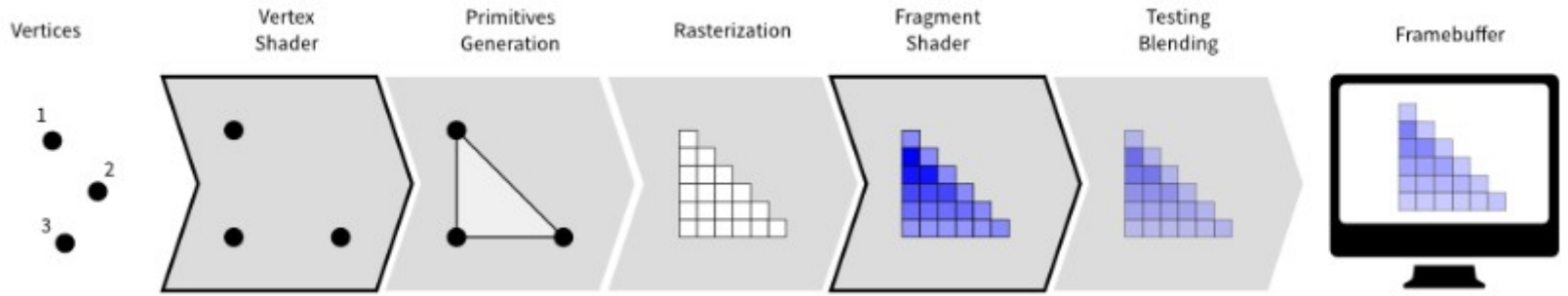

OpenGL 4.4 draw dispatch
Conditional Rendering (10.10)
Query Object (2.5.12)
Skip draw
indirect
direct
Compute Shading (19)
gl_NumWorkGroups
gl_WorkGroupSize
gl_WorkGroupID
gl_LocalInvocationID
gl_GlobalInvocationID
gl_LocalInvocationIndex
Indirect Draw Buffer (10.3.10)
Draw Dispatching (10.5)
Draw (10.5)
Primitive Mode (10.1)
OpenGL 4.4 vertex processing
Draw (10.5)
Vertex Invocation (10.5)
First
BaseVertex
BaseInstance
gl_InstanceID (10.3.3)
Vertex Assembly (10.8)
Array Buffer (10.3.8)
Vertex Array Object (10.4)
gl_VertexID (11.1.3.9)
Fetch Vertex Cache
Not specified but expected from most hardware
Cached
Element Array Buffer (10.3.9)
Vertex Shading (11.1)
gl_Position (11.1.3.10)
gl_PointSize (11.1.3.10)
gl_ClipDistance (11.1.3.10)
Vertex
OpenGL 4.4 Tessellation
gl_Position (11.2.1.2.2)
gl_PointSize (11.2.1.2.2)
gl_ClipDistance (11.2.1.2.2)
Primitive Assembly (14)
gl_PrimitiveID (11.2.1.2.2)
gl_InvocationID (11.2.1.2.2)
gl_PatchVerticesIn (11.2.1.2.2)
gl_PerVertex
Patch
Tess Control Shading (11.2.1)
Number of control shader invocations and number of vertices in the output patch.
gl_PrimitiveID
vertices
Per-patch data (11.2.1.2.3)
gl_TessLevelOuter
gl_TessLevelInner
Primitive Assembly (14)
Primitive Generation (11.2.2)
cull
gl_TessCoord (11.2.3.3)
Tess Evaluation Shading (11.2.3)
Primitive Mode (11.2.2)
Ordering (11.2.2)
Vertex Spacing (11.2.2)
gl_Position (11.2.3.4)
gl_PointSize (11.2.3.4)
gl_ClipDistance (11.2.3.4)
Tessellate


OpenGL 4.4 Primitive Processing
Transform Feedback Object (2.5.11)
Vertex
gl_Position
gl_PointSize
gl_ClipDistance
Input Primitive Mode (11.3.1)
Primitive Assembly (14)
Primitive
Geometry Shading (11.3)
gl_InstanceID
Primitive Instancing (11.3.4.2)
Instancing
Primitives = 0
The shader setup override the fixed pipeline state.
gl_PrimitiveIDIn
max_vertices
Invocations
Output Primitive Mode (11.3.2)
Primitive Assembly (14)
Transform Feedback (13.2)
rasterization
transform feedback
Stream (11.3.4.3)
Transform Feedback Buffer (2.5.11)
Compute Shading (19)
Transform Feedback Buffer (2.5.11)
Transform Feedback Primitives Written (13.3)
stream_id > 0
stream_id = 0
Point
Primitive
Query Object (2.5.12)
Flat Shading (13.4)
flat
smooth
Perspective Division (13.6)
no perspective
Controlling the Viewport (13.6.1)
Primitive Clipping (13.5)
clipped
gl_ClipDistance
gl_PrimitiveID
gl_ViewportIndex
gl_Layer

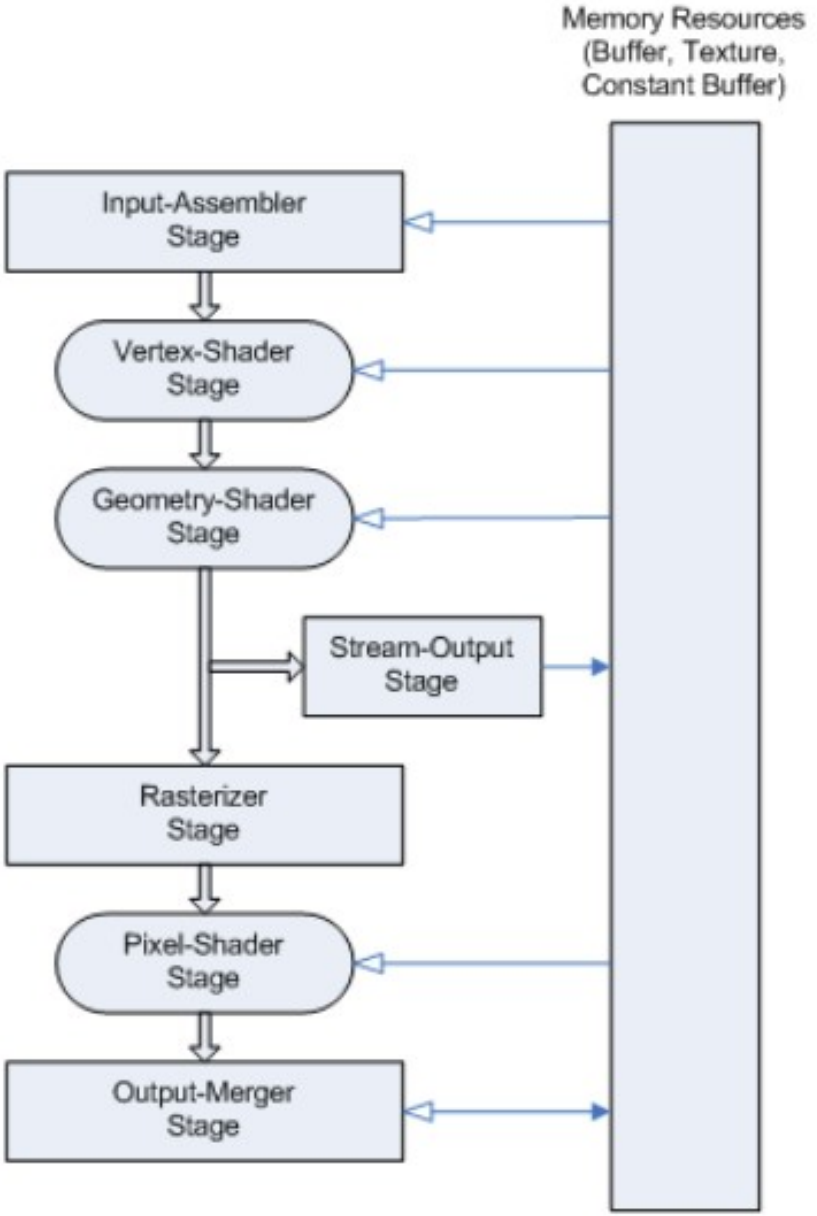
Cry Engine 3 (zródło: CryTek)


Vertex specification
format, kolejność, instancing
Vertex shader
Tesselation
Compute shader
Geometry shader
Rasterization
clipping, culling
Fragment shader
Per sample operations
scissor test, alpha test, stencil test, depth test, blending, konwersje



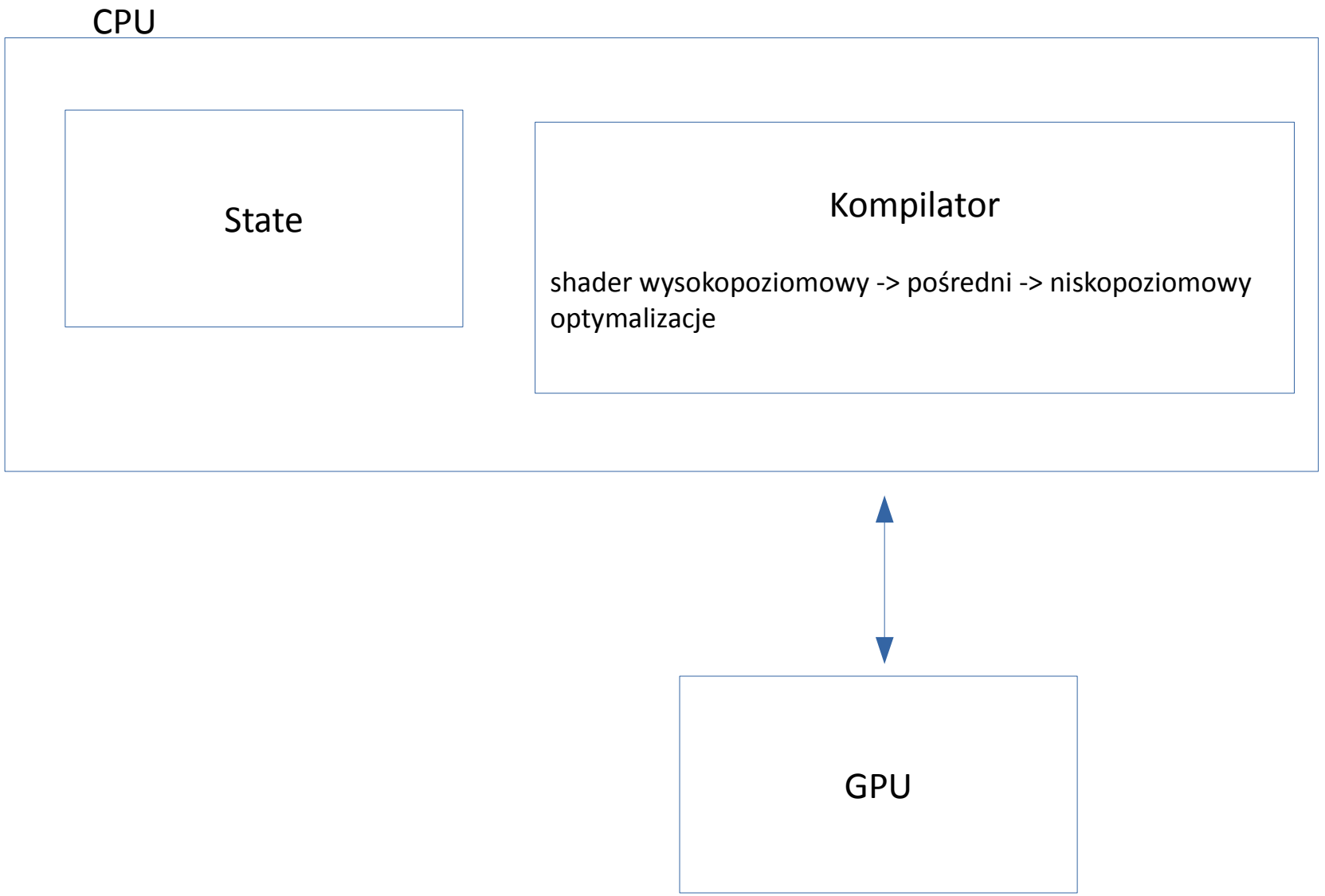


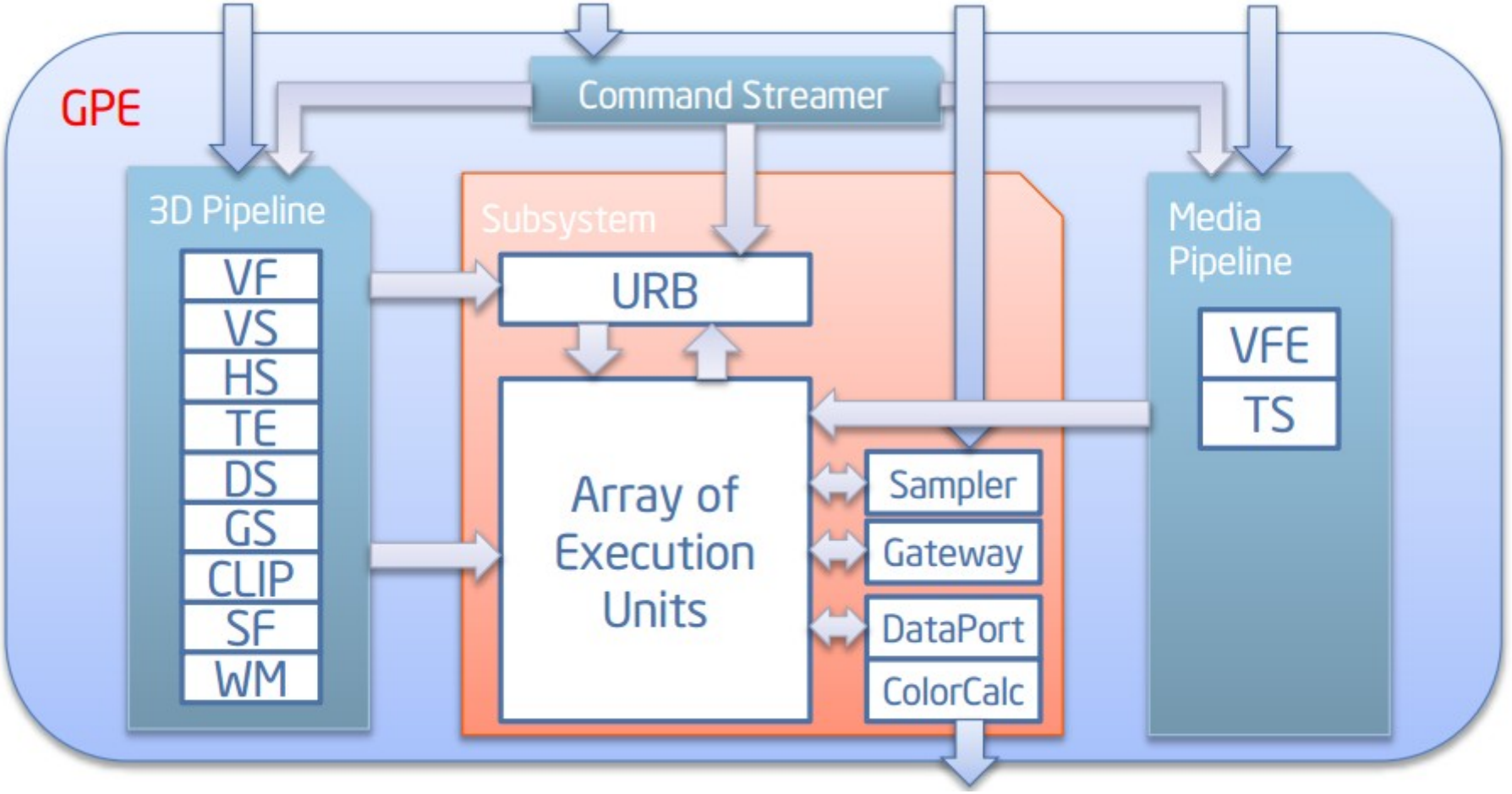
Źródło obrazka: https://glumpy.github.io/modern-gl.html



Wysokopoziomowy potok przetwarzania w DirectX 10 (źródło: DirectX SDK)

Driver


CPU
State
Kompilator
shader wysokopoziomowy -> pośredni -> niskopoziomowy
optymalizacje
GPU

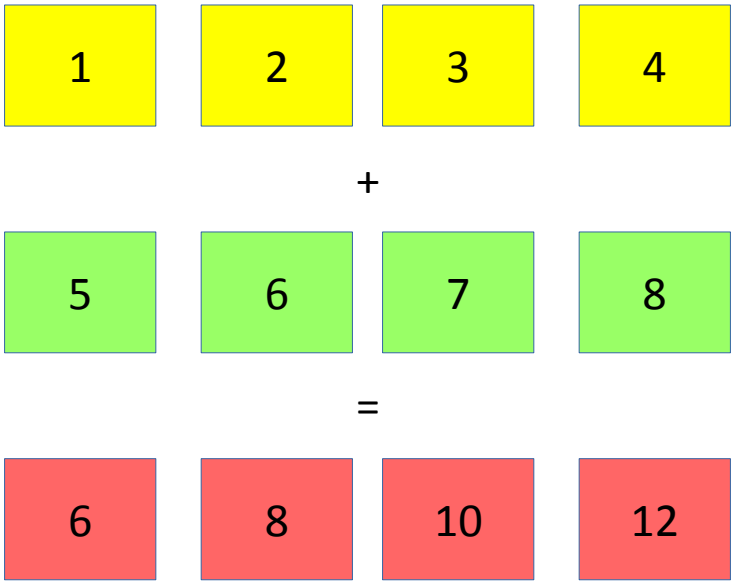


Źródło: intel.com

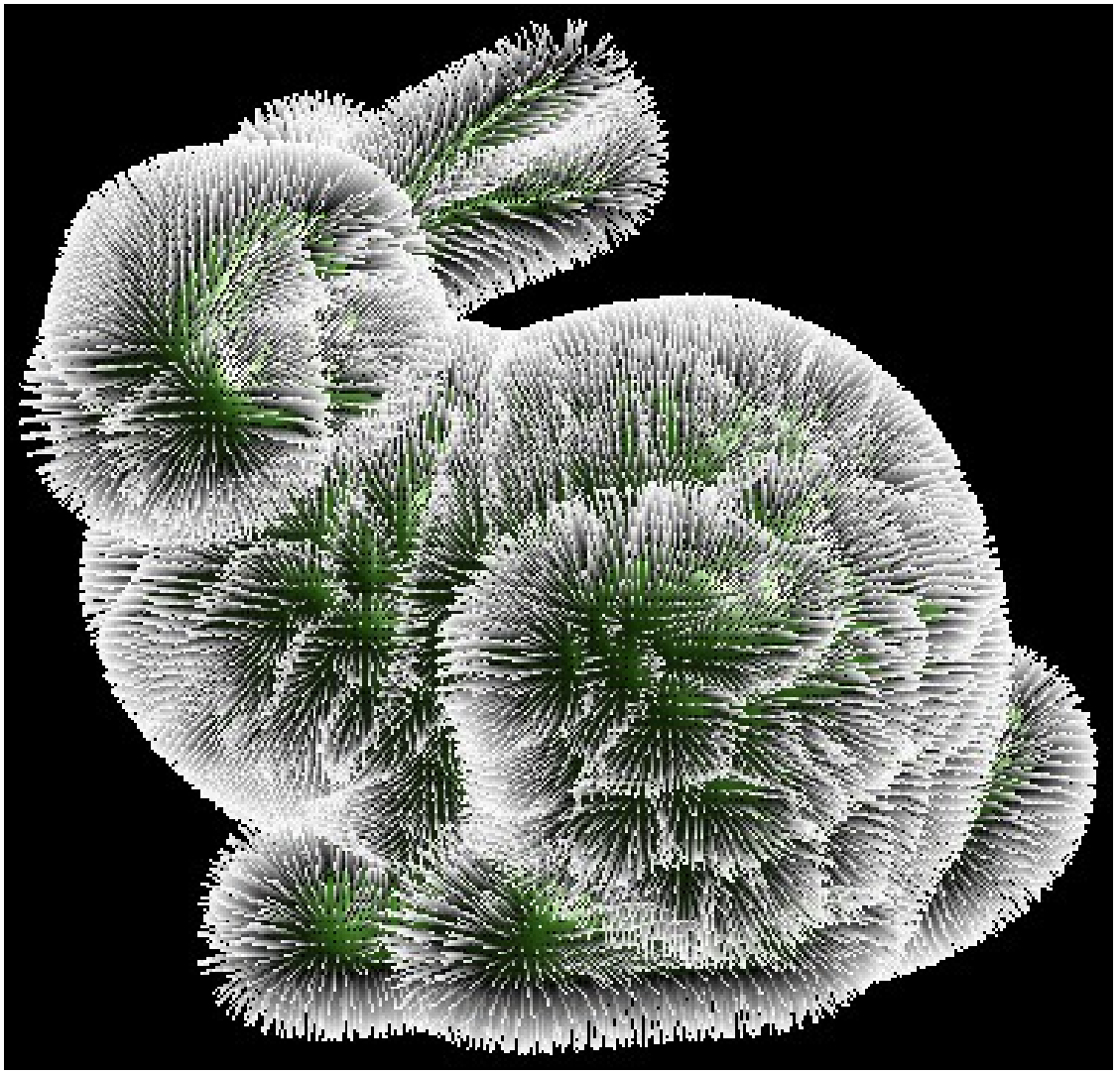
SIMD

Single **I**nstruction **S**ingle **D**ata

Single **I**nstruction Multiple **D**ata

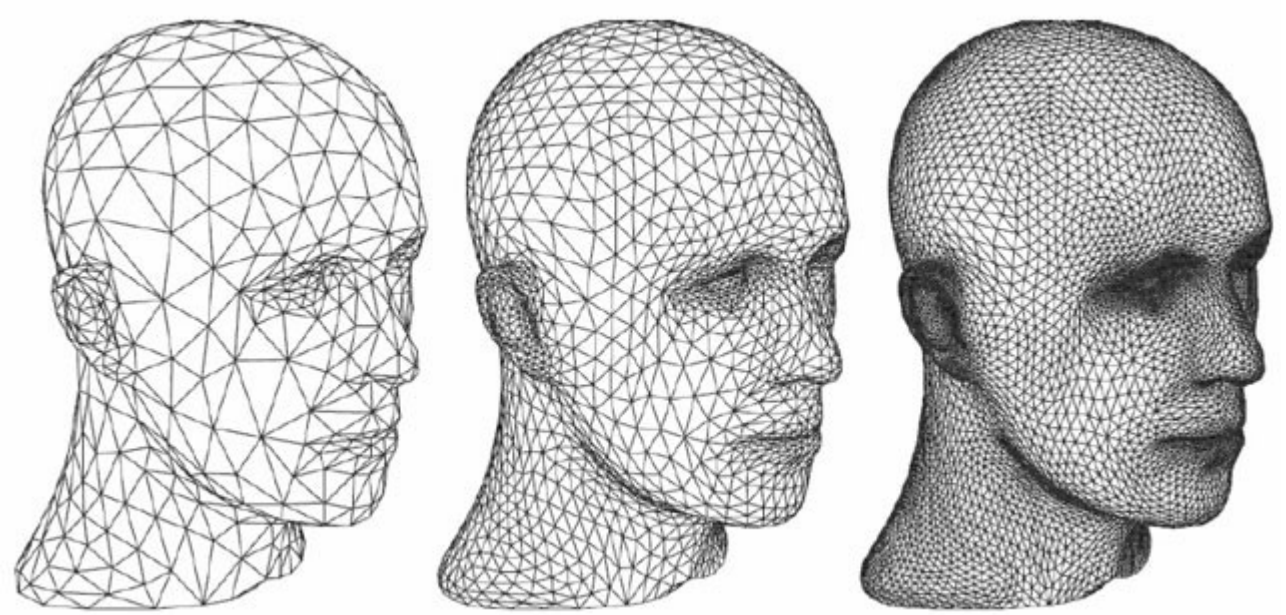


Geometry shader



źródło: https://bassemtodary.wordpress.com

Teselacja




Tessellation Off


DX11 Tessellation On

3. Shadery

Przykład GLSL

```
void main()  // vertex
{
    gl_Position = vec4( 0.0, 0.0, 0.0, 1.0 );
}


void main()  // fragment
{
    gl_FragColor = vec4( 0.0, 0.0, 0.0, 1.0 );
}
```

Przykład HLSL

Wierzchołek
-
pozycja w przestrzeni, kolor,
normalna, koordynaty tekstury, ...

```
// parametry dla każdego wierzchołka
struct VertexShader_input
{
    float4 position: POSITION;
    float4 color: COLOR;
}
// kolor wynikowy
struct VertexShader_output
{
    // przetransformowana do układu świat-widok-projekcja
    float4 position: POSITION;
}

void VS_Forward(VertexShader_input input, VertexShader_output output)
{
    VertexShader_output out;
    out.position = mul (input.position, wvp_matrix);

    float diffuse = max(0, dot( -light_dir, input.normal));
    out.color = input.color * diffuse;
}
```

Assembler

```
float x=(a*abs(f))>(b*abs(f));
```

HLSL

```
abs r7.w, c2.x
mul r2.w, r7.w, c0.x
mul r9.w, r7.w, c1.x
slt r0.w, r9.w, r2.w
```

Assembly

```
02000023 80080007 a0000002
03000005 80080002 80ff0007 a0000000
03000005 80080009 80ff0007 a0000001
0300000c d00f0000 80ff0009 80ff0002
```

DX p-code

```
983a28f41595
329d8d123c04
329d8d627c08
3b487794333a
```

HW microcode

Assembler

```
// deklaracja rodzaju shadera, vs wskazuje na Vertex Shader
vs_4_0 // druga część na wersję (4.0)
// deklaracja tablicy stałych (uniforms)
dcl_constantbuffer  cb0[11], immediateIndexed
// deklaracja wartości zmiennych dla każdego wierzhołka
dcl_input v0.xyzw
dcl_input v1.xyz
// wartości zwracane
dcl_output_siv  o0.xyzw , position
dcl_output o1.xyzw
// rejestry tymczasowe
dcl_temps 1
// wykonuje mnożenie macierzy przez wektor
dp4 o0.x, v0.xyzw, cb0[0].xyzw
dp4 o0.y, v0.xyzw, cb0[1].xyzw
dp4 o0.z, v0.xyzw, cb0[2].xyzw
dp4 o0.w, v0.xyzw, cb0[3].xyzw
// negacja znaku
mov r0.xyz, -cb0[10].xyzx
// iloczyn skalarny
dp3 r0.x, r0.xyzx, v1.xyzx
// ogranicza wielkość od dołu (0)
max r0.x, r0.x, l(0.000000)
// mnoży wektor przez stałą
mul o1.xyzw, r0.xxxx, l(1.000000, 0.000000, 1.000000, 0.500000)
ret // koniec shadera, wartość końcowa w rejestrze o1
```

```
uniform float scale;

attribute vec2 position;
attribute vec4 color;
varying vec4   v_color;

void main()
{
    gl_Position = vec4(position*scale, 0.0, 1.0);
    v_color = color;
}
```

```
varying vec4 v_color;

void main()
{
    gl_FragColor = v_color;
}
```

API

```
#include <GL/glew.h>
#include <GLFW/glfw3.h>



GLuint vs;
vs = glCreateShader(GL_VERTEX_SHADER);

const char* source = myLoadShader(filename);

glShaderSource(vs, 1, &source, NULL);
glCompileShader(vs);
```

```
if (!vs)
{
    printf("ERROR Could not compile the shader");
    return 0;
}

GLint status;
glGetShaderiv( vs, GL_COMPILE_STATUS, &status );




if (status == GL_FALSE)
{
     GLint infoLogLength;
     glGetShaderiv( shader, GL_INFO_LOG_LENGTH, &infoLogLength );

     GLchar* strInfoLog = new GLchar[infoLogLength + 1];
     glGetShaderInfoLog( shader, infoLogLength, NULL, strInfoLog );

     fprintf( stderr, "Compilation error in shader %s\n", strInfoLog );
     delete[] strInfoLog;
}
```

```
GLuint shader_program = glCreateProgram();

glAttachShader(shader_program, vs);
glAttachShader(shader_program, fs);

glLinkProgram(shader_program);
int result = -1;
glGetProgramiv(shader_program, GL_LINK_STATUS, &result);


if (GL_TRUE != result)
{
        GLint infoLogLength;
        glGetProgramiv( shader_program, GL_INFO_LOG_LENGTH, &infoLogLength );
        glGetProgramInfoLog( shader_program, infoLogLength, NULL, strInfoLog );
       // ...
}
```

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

glUseProgram( shader_program );
glBindVertexArray( vao );


glBindTexture( GL_TEXTURE_2D, tex2d );
glDrawArrays( GL_TRIANGLES, 0, 3 );


glfwSwapBuffers( window );
glfwPollEvents();
```

Instrukcje

- arytmetyczne (*np. mov, mul, add*), logiczne (*and, or, xor*)
- wektorowe (*dp3, dp4*)
- matematyczne (*sin, log2*)
- warunkowe (*if, else*)
- pętle (*for, while, break, continue*)
- samplowanie
- zapisy do buforów/tekstur (*writert, store, atomic*)
- *discard*

Dane

- wektory, macierze
- liczby zmiennoprzecinkowe, całkowite, logiczne
- tablice
- struktury
- tekstury, bufory

```
vec2 a = vec2(1.0,2.0);

bvec3

mat2, mat3, mat4

struct dirlight
{
    vec3 direction;
    vec3 color;
};
```

Kwalifikatory

const – stała czasu kompilacji

attribute – globalne zmienne które mogą się zmieniać per vertex (np. kolor czy texcoord), wysyłane z aplikacji do vertex shaderów. Dostępny tylko w VS, tylko do odczytu.

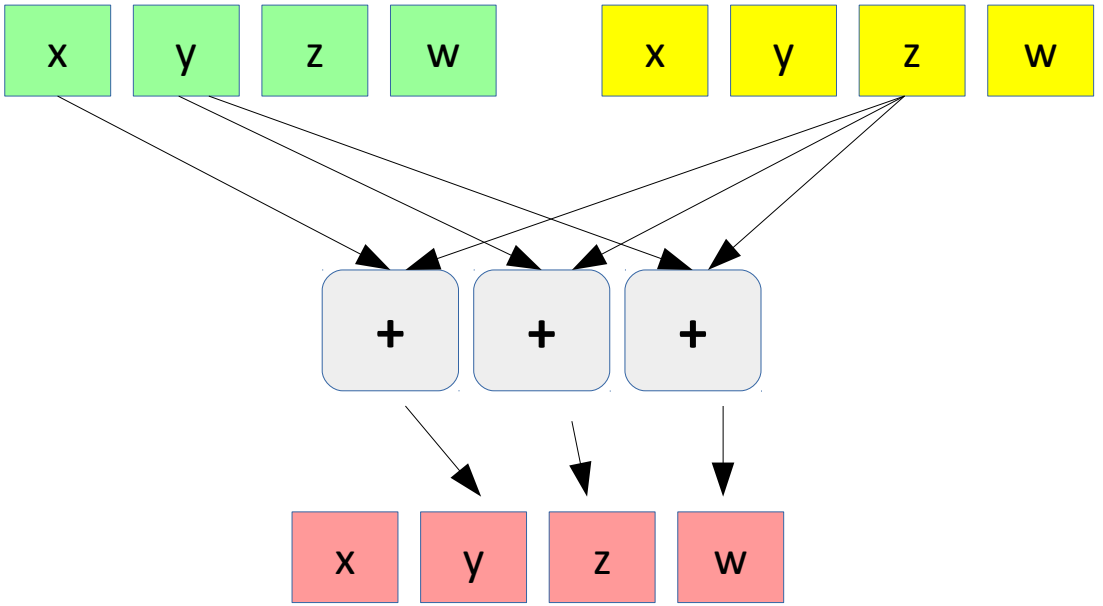
uniform – zmienna globalna która może się różnić per-primitive, wysyłana z aplikacji do shaderów. Dostępne w VS i PS, tylko do odczytu.

varying – używane do przesyłania zinterpolowanych danych do pixel shadera. Można zapisać w VS, tylko do odczytu w PS.

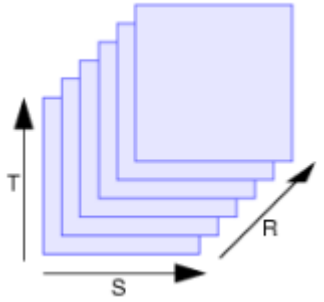
Swizzle i writemaski

```
.r,   .rrrr,   .xxxx,   .x
.g,   .gggg,   .yyyy,   .y
.b,   .bbbb,   .zzzz,   .z
.a,   .aaaa,   .wwww,   .w
```

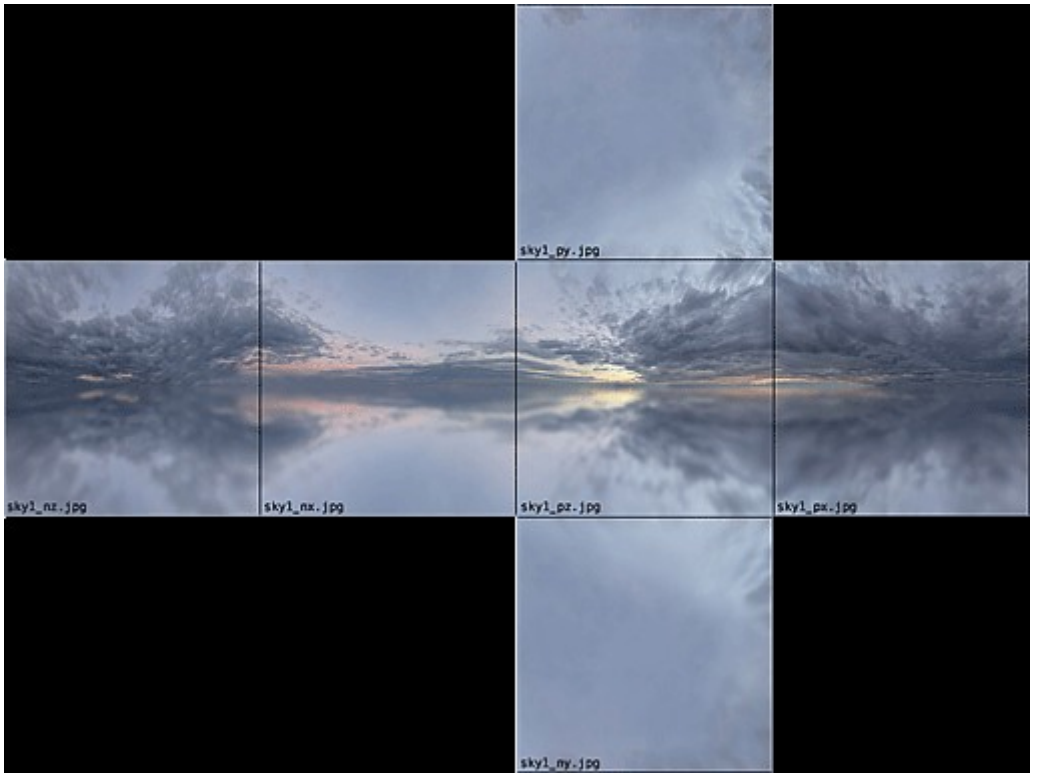
```
a.yzw = b.xyy + c.z
```



Tekstury i samplery



- 1D
- 2D
- 3D
- cube
- tablice powyższych



sampler1D, 2D, 3D
samplerCube
sampler1DShadow

```
#version 430
#extension GL_ARB_shader_image_load_store : enable

out vec4 frag_color;
in vec3 color;

writeonly layout(rgba32f) uniform imageCube write_tex;


void main()
{
  frag_color = vec4( 0.2, 0, 0, 1);

  for (int i = 0; i < 6; ++i)
  {
      imageStore(write_tex, ivec3(0, 0, i), vec4( i / 6.0) );
  }

}
```

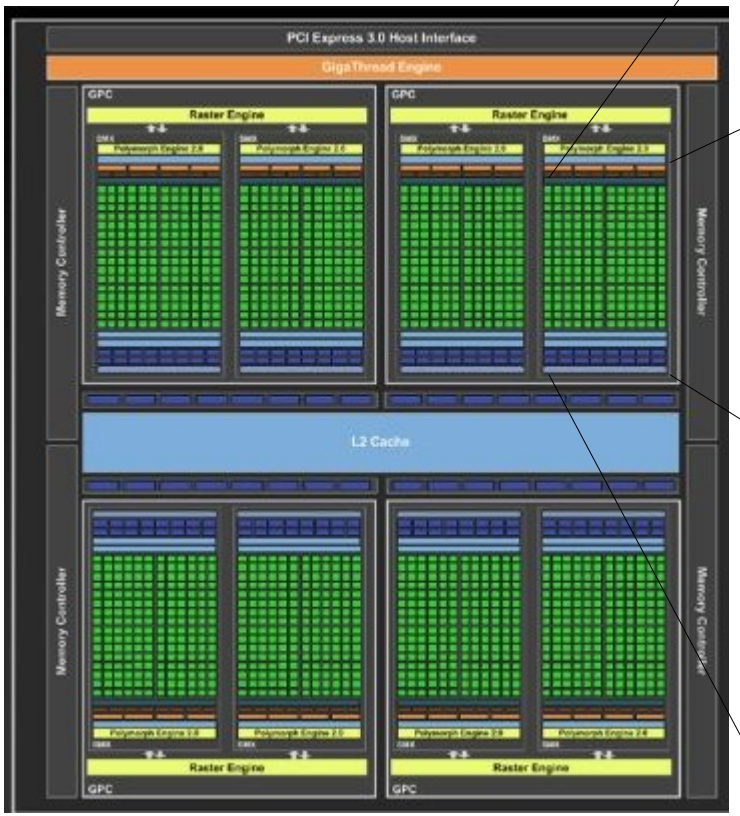
4. Zastosowania shaderów

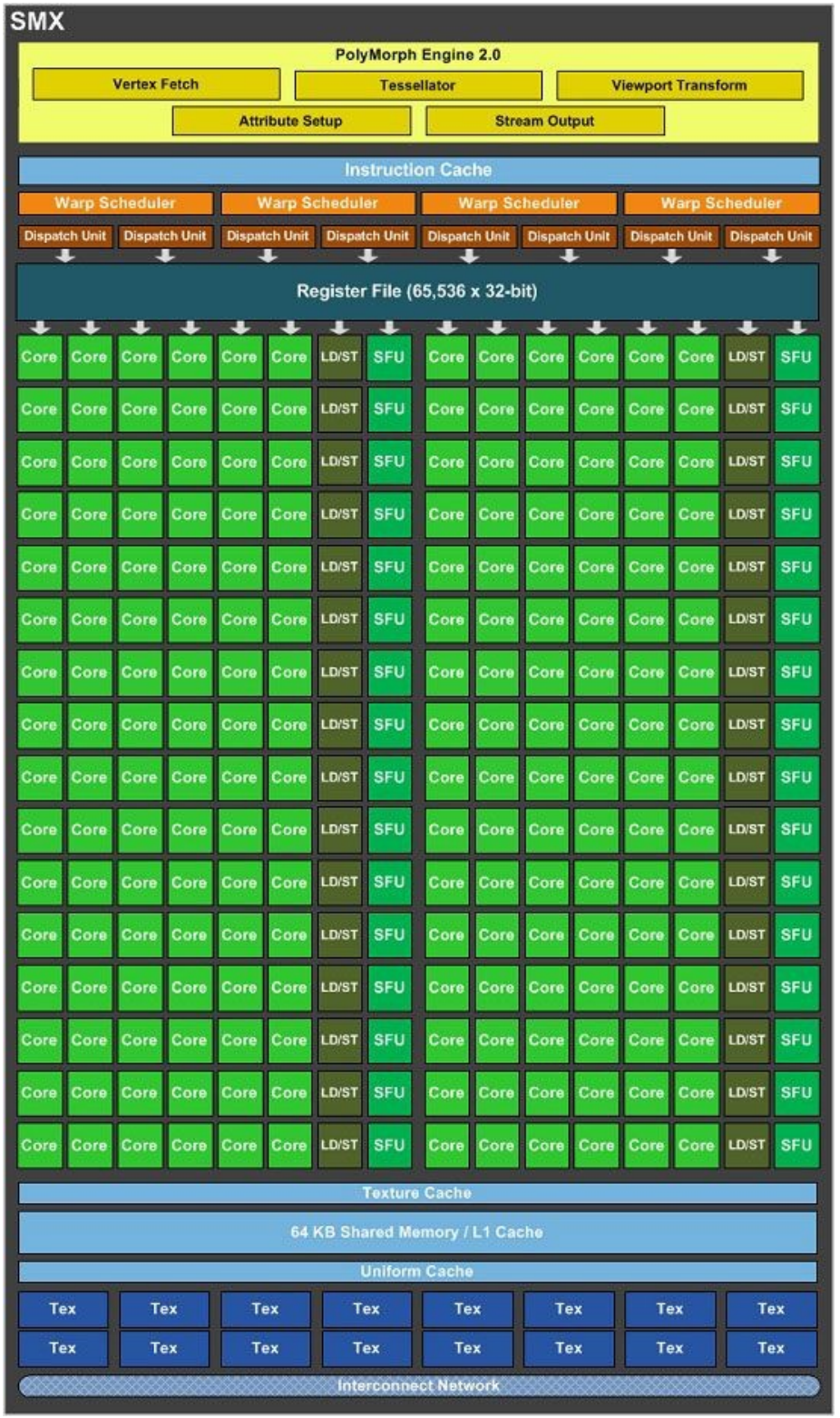
Grafika 3d

źródło: https://bassemtodary.wordpress.com

GPGPU

nVidia Kepler 1536 CUDA cores





```
// Input and output grids have 10000 x 10000 or 100 million elements.

void transform_10k_by_10k_grid(float in[10000][10000], float out[10000][10000])
{
    for (int x = 0; x < 10000; x++) {
        for (int y = 0; y < 10000; y++) {
            // The next line is executed 100 million times
            out[x][y] = do_some_hard_work(in[x][y]);
        }
    }
}
```

Źródło: wikipedia

```
kernel void transform_grid( global const float ** in, global float ** out)
{
    int a = get_global_id(0);
    int b = get_global_id(1);
    out[a][b] = do_some_hard_work( in[a][b] );
}
```

NVIDIA DRIVE PX 2

12 CPU cores | Pascal GPU | 8 TFLOPS | 24 DL TOPS | 16nm FF | 250W | Liquid Cooled

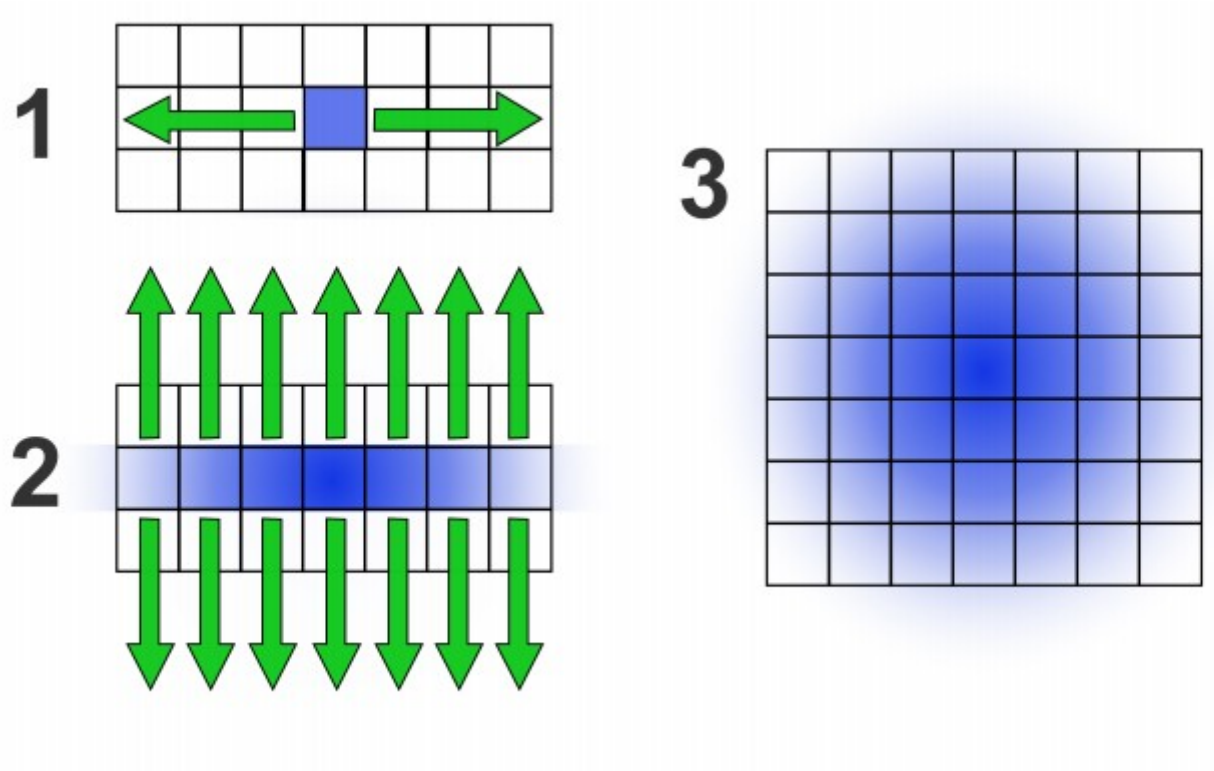
5. Przykładowe zastosowania

5.1 Bloom

zródło: www.grasz.org/recenzje/two-worlds-,1,1,2643

Separowalne rozmycie Gaussowskie

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$



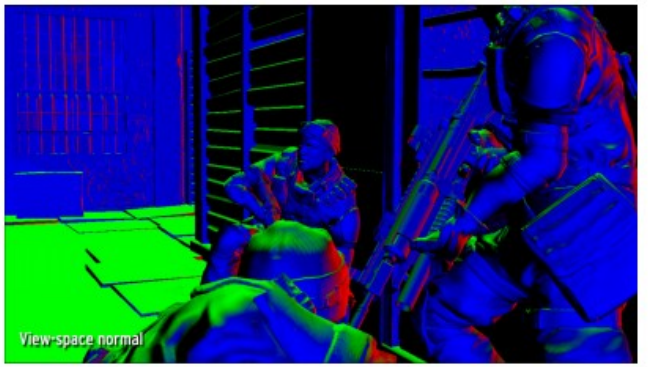
```
Texture2D diffuse;

static const int SAMPLE_COUNT = 15;

// tablice wag i przesunięć
float sample_weights[SAMPLE_COUNT];
float2 sample_offsets[SAMPLE_COUNT];

float4 PS_flat( in PS_flat_input _in) : SV_Target
{
    float4 color = 0;

    for( int i = 0; i < SAMPLE_COUNT; ++i )
    {
        // dodaj wpływ każdego piksela sąsiedztwa (oddalonego o
           przesunięcie), bazując na jego wadze
        float4 sampled = diffuse.Sample( sampler_linear, _in.texcoord +
           sample_offsets[i] );
        color += sampled * sample_weights[i];
    }

    return color;
}
```

(a) **Bez blooma**

(b) **Z bloomem**

5.2 Deferred shading

(a) **Głębokość**



(b) **Normalne w przestrzeni widoku**

(c) **Albedo**

(d) **Pole wektorowe ruchu**

(e) **Miękkość rozbłysku**

(f) **Intensywność rozbłysku**

(a) **Podstawowe połączenie G-Bufora**

(b) **Finalny obraz z post processingiem** (**głębia widzenia, bloom, rozmycie ruchu, korekcja kolorów**)

Deferred Rendering in Killzone 2. 2005. http://www.guerrilla-games.com/publications/dr_kz2_rsx_dev07.pdf

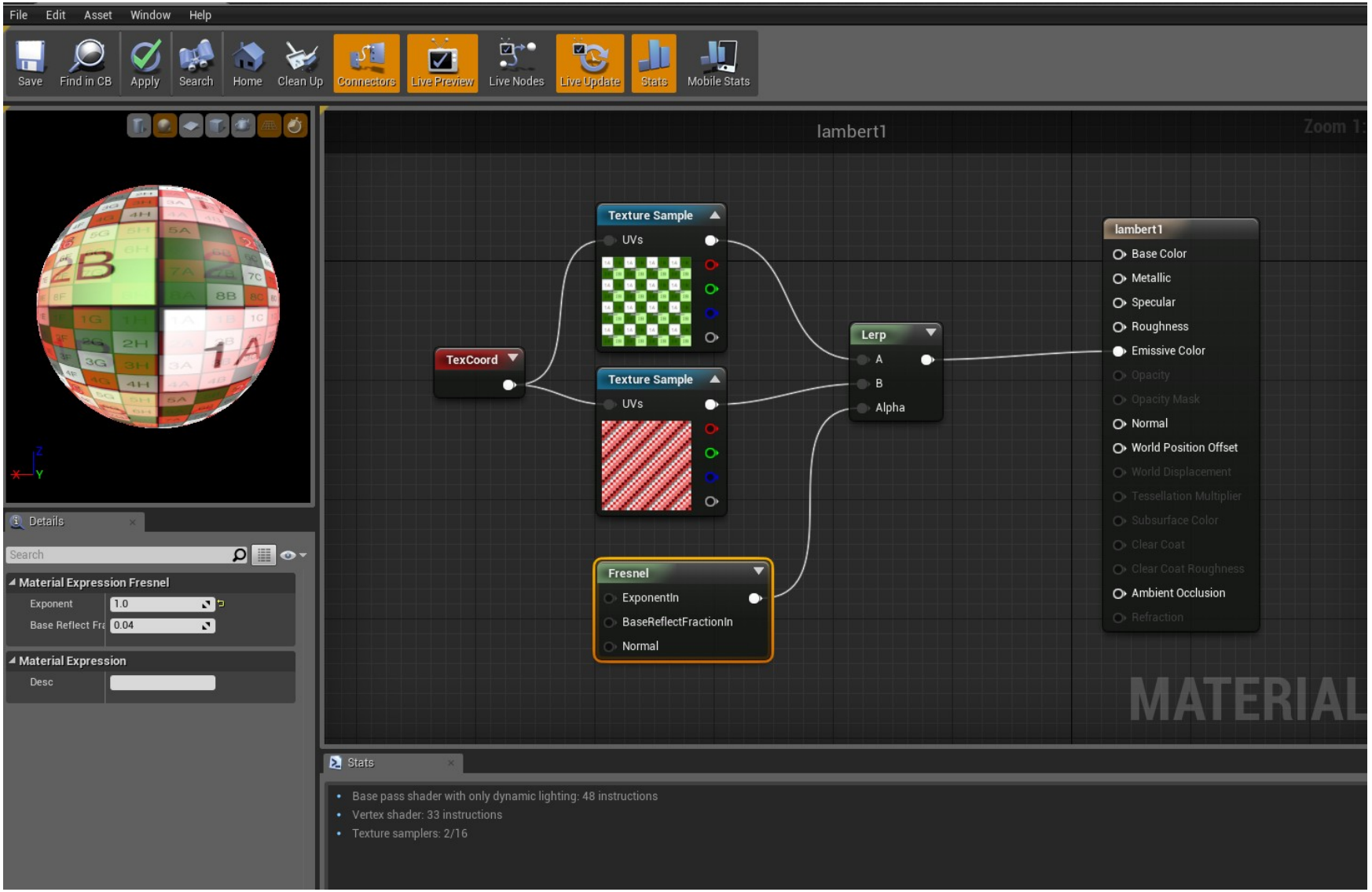
```
1  struct Vertex_in
2  {
3      float4 position : POSITION;
4      float3 normal   : NORMAL;
5      float2 texcoord : TEXCOORD0;
6  };
7
8  struct Vertex_out
9  {
10     float4 position   : SV_POSITION; // przestrzeń wvp
11     float3 position_w : TEXCOORD0;   // przestrzeń świata
12     float3 normal     : TEXCOORD1;
13     float2 texcoord   : TEXCOORD2;
14 };
15
16 void VS_FillGbuffer(  in    Vertex_in  _in,
17                       out   Vertex_out _out )
18 {
19     // przekształca pozycję wierzchołka do przestrzeni wvp
20     // aby umieścić model w odpowiednim miejscu dla potoku
21     _out.position   = mul( _in.position, wvp_matrix );
22     // natomiast do G-Bufora zapisana zostanie pozycja w przestrzeni
          świata
23     _out.position_w = mul( _in.position, world_matrix).xyz;
24
25     // przekształć normalną do przestrzeni swiata
26     // 0 jest potrzebne do tego aby zignorować translację w macierzy
27     _out.normal     = mul( float4( _in.normal, 0), world_matrix);
28     _out.texcoord   = _in.texcoord;
29 }
```

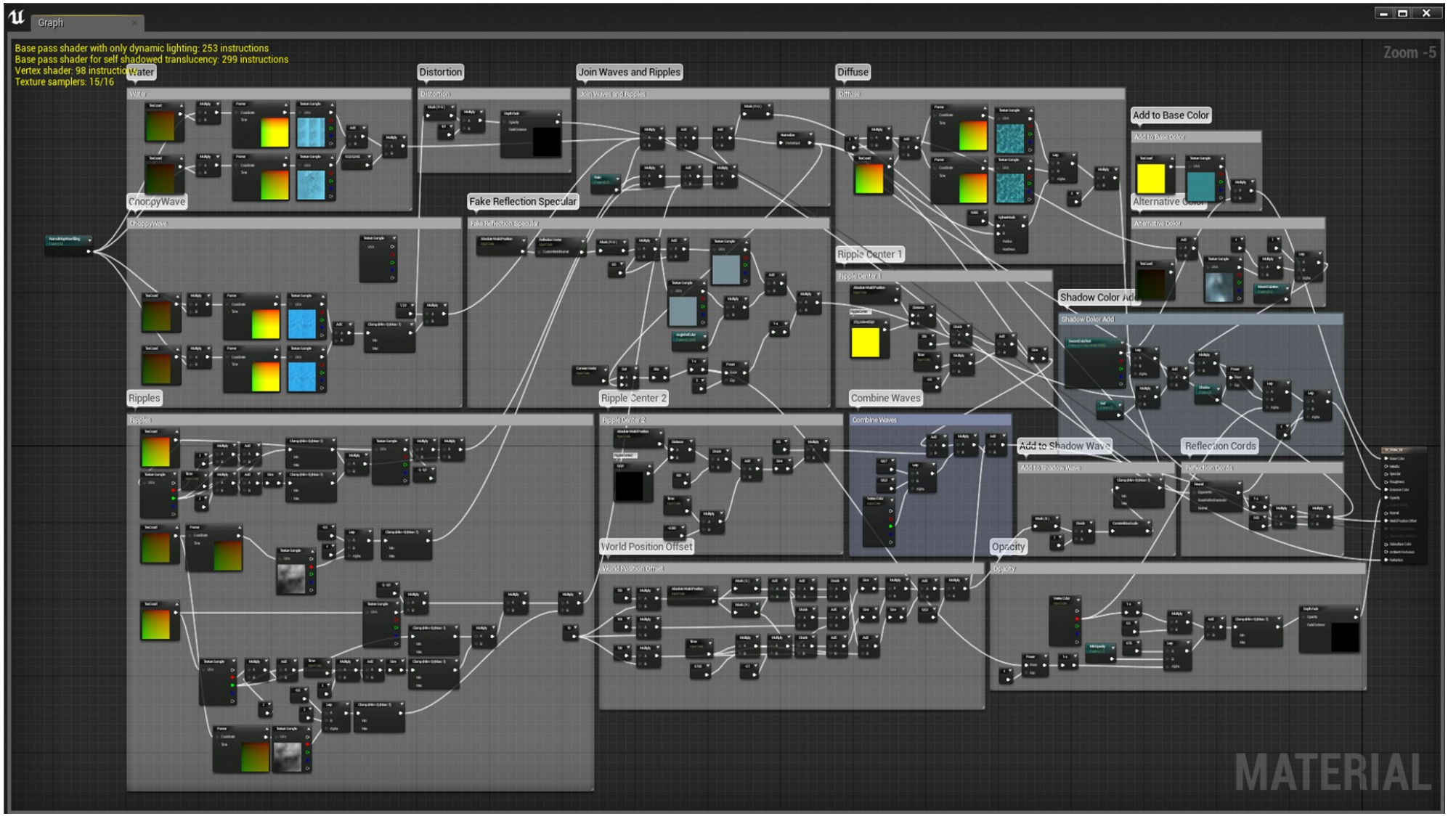
```
// struktura określa wektory wyjściowe dla 3 render targetów
// zamiast jednego standardowego SV_Target0
struct Pixel_out
{
    float4 position : SV_Target0;
    float4 normal   : SV_Target1;
    float4 albedo   : SV_Target2;
};

void PS_FillGbuffer( in Vertex_out _in,
                     out Pixel_out gbuffer )
{
   gbuffer.position = float4( _in.position_w, 1);

   // znormalizuj normalną, mogła przestać nią być podczas
   // interpolacji po powierzchni trójkąta
   gbuffer.normal = float4( normalize(_in.normal), 1);
   gbuffer.albedo = texture_diffuse.Sample(sampler_linear,_in.texcoord);
}
```

Deferred Rendering - Interactive 3D Graphics

https://www.youtube.com/watch?v=nSL8cOxtsz4

Ręczne pisanie shaderów?




Graph
Base pass shader with only dynamic lighting: 253 instructions
Base pass shader for self shadowed translucency: 299 instructions
Vertex shader: 98 instructions
Texture samplers: 15/16
Zoom -5
Water
Distortion
Join Waves and Ripples
Diffuse
Add to Base Color
Alternative Color
ChoppyWave
Fake Reflection Specular
Ripple Center 1
Shadow Color Add
Ripples
Ripple Center 2
Combine Waves
Add to Shadow Wave
Reflection Cords
World Position Offset
Opacity
MATERIAL

Insomnium Engine. Deferred Rendering. Physically based shading and post processing effects

https://www.youtube.com/watch?v=bj9P504JUAE

Koniec