Zaawansowane systemy programowania grafiki. Teselacja

Aleksander Denisiuk
Uniwersytet Warmińsko-Mazurski
Olsztyn, ul. Słoneczna 54
denisjuk@matman.uwm.edu.pl

16 marca 2021

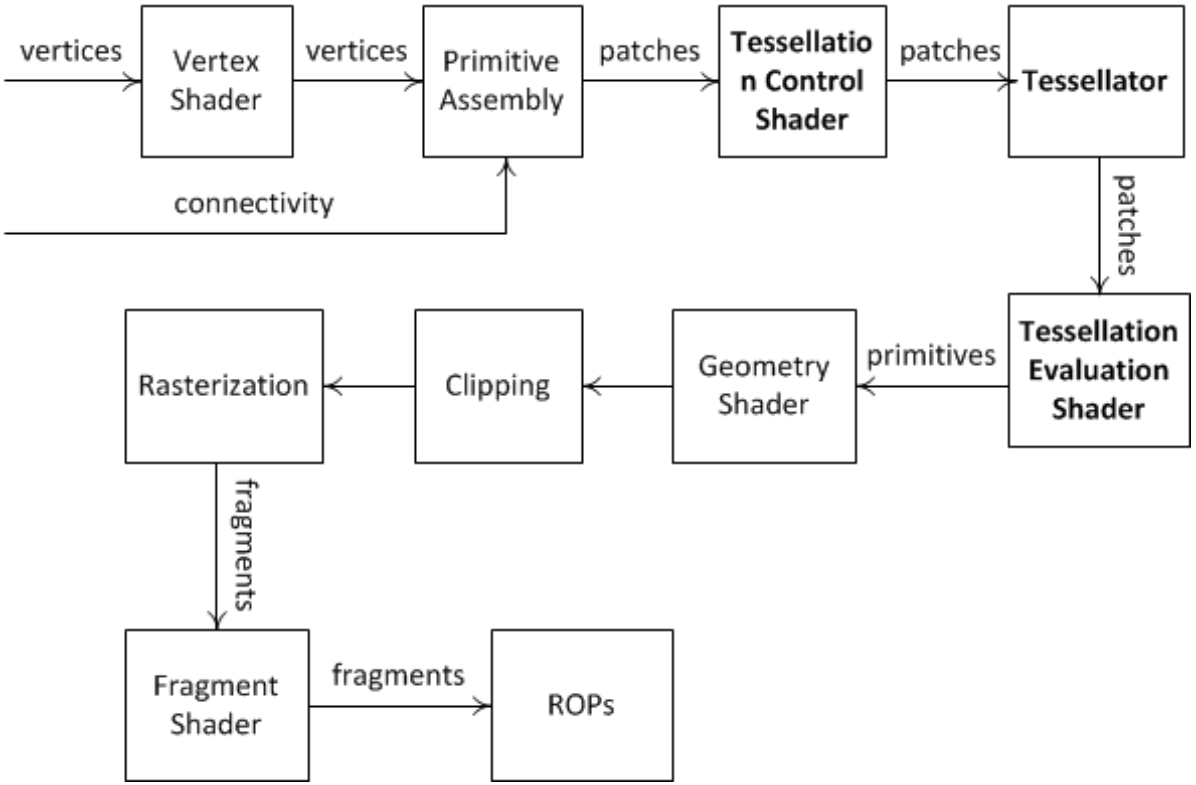
Teselacja

Najnowsza wersja tego dokumentu dostępna jest pod adresem

http://wmii.uwm.edu.pl/~denisjuk/uwm

Teselacja

Potok renderingu 4



Nowy tryb

- `GL_PATCHES`
- prykład:

```
glDrawArrays(GL_PATCHES, 0, 6);
```

- nowy prymityw graficzny zawiera dowolną ilość wierzchołków (na przykład, punkty kontrolne NURBS)

Ilość wierzchołków

- w shaderze kontroli teselacji:

```
layout (vertices = 4) out;
```

- w kodzie programu:

```
glPatchParameteri(GL_PATCH_VERTICES, 4);
```

Shader kontroli teselacji

- `GL_TESS_CONTROL_SHADER`
- dla każdego wierzchołka każdego `patch`a
- ma dostęp do wszystkich wierzchołków
- określa parametry dla teselatora oraz shadera ewaluacji teselacji

Parametry wejściowe

- jako tabele

```
in vec3 normal [];
```

- parametry wbudowane
  - `gl_in[]` — tablica struktur
    - `gl_Position`
    - `gl_PointSize`
    - `gl_ClipDistance[]`
  - `gl_PatchVerticesIn` — ilość wierzchołków
  - `gl_PrimitiveID` — numer prymitywu
  - `gl_InvocationID` — numer wierzchołka w prymitywie

Zmienne uniform i tekstury

- shader ma dostęp do zmiennych uniform i do tekstur

Dane wyjściowe

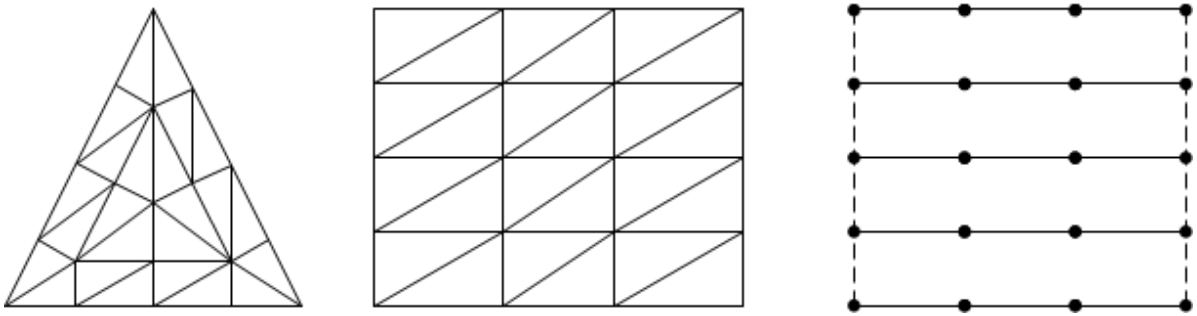
- `gl_out[]` — dane wierzchołków
  - `gl_Position`
  - `gl_PointSize`
  - `gl_ClipDistance[]`
- `gl_TessLevelOuter[]` — rozbicie granic prymitywu
- `gl_TessLevelInner[]` — rozbicie wnętrza prymitywu
- inne dane wyjściowe, opisane jako tabele, dla każdego wierzchołka
- zmienne typy `patch`

```
patch out float my_value
```

Funkcja `barrier()`

- opracowanie wierzchołków jest równoległe, kolejność nie określona
- synchronizacja: `barrier()`

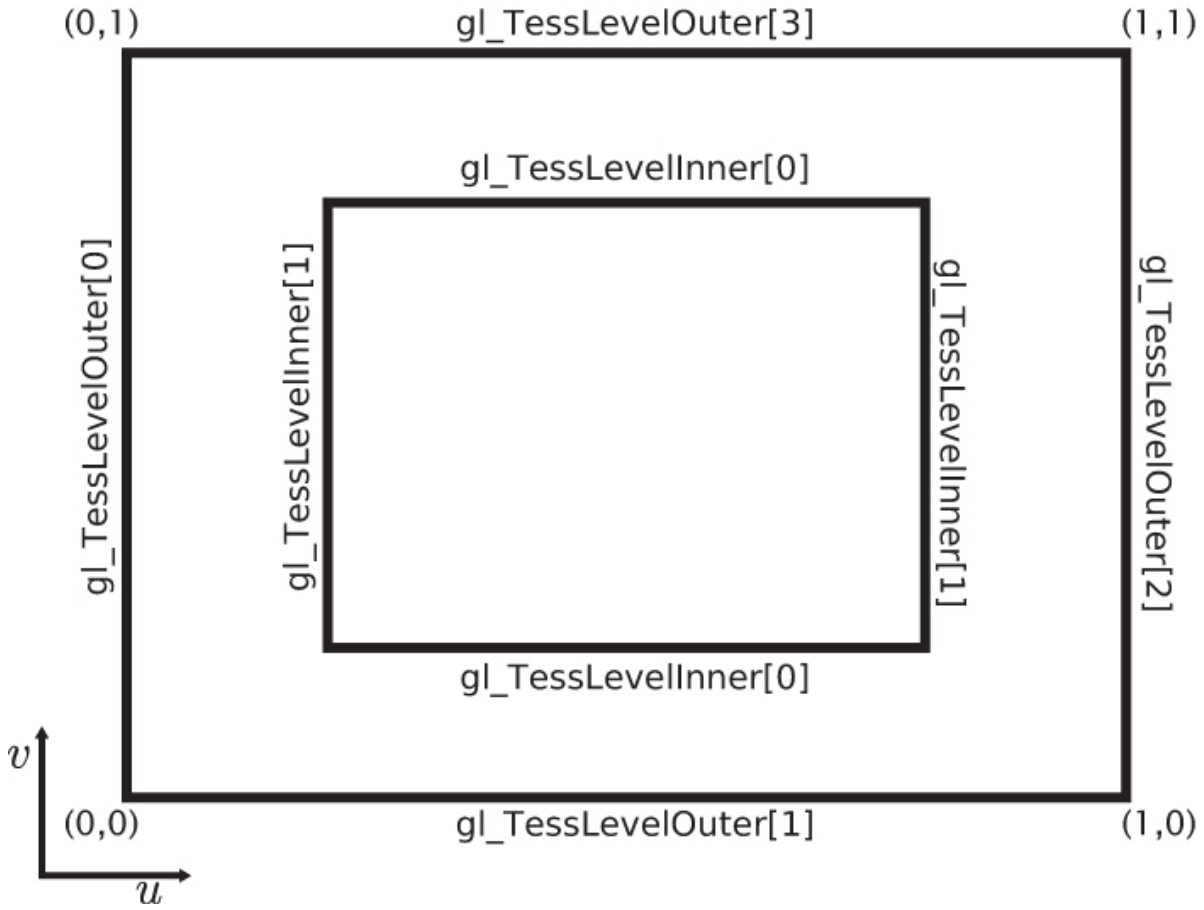
Shader ewaluacji teselacji



- określa się typ: (`triangles`, `quads` bądź `isolines`)
- sposób podziału odcinków (`equal_spacing`, `fractional_even_spacing` bądź `fractional_odd_spacing`)
- orientacj `cw` bądź `ccw`

```
layout (triangles, equal_spacing, ccw) in;
```

Teselacja w trybie quads



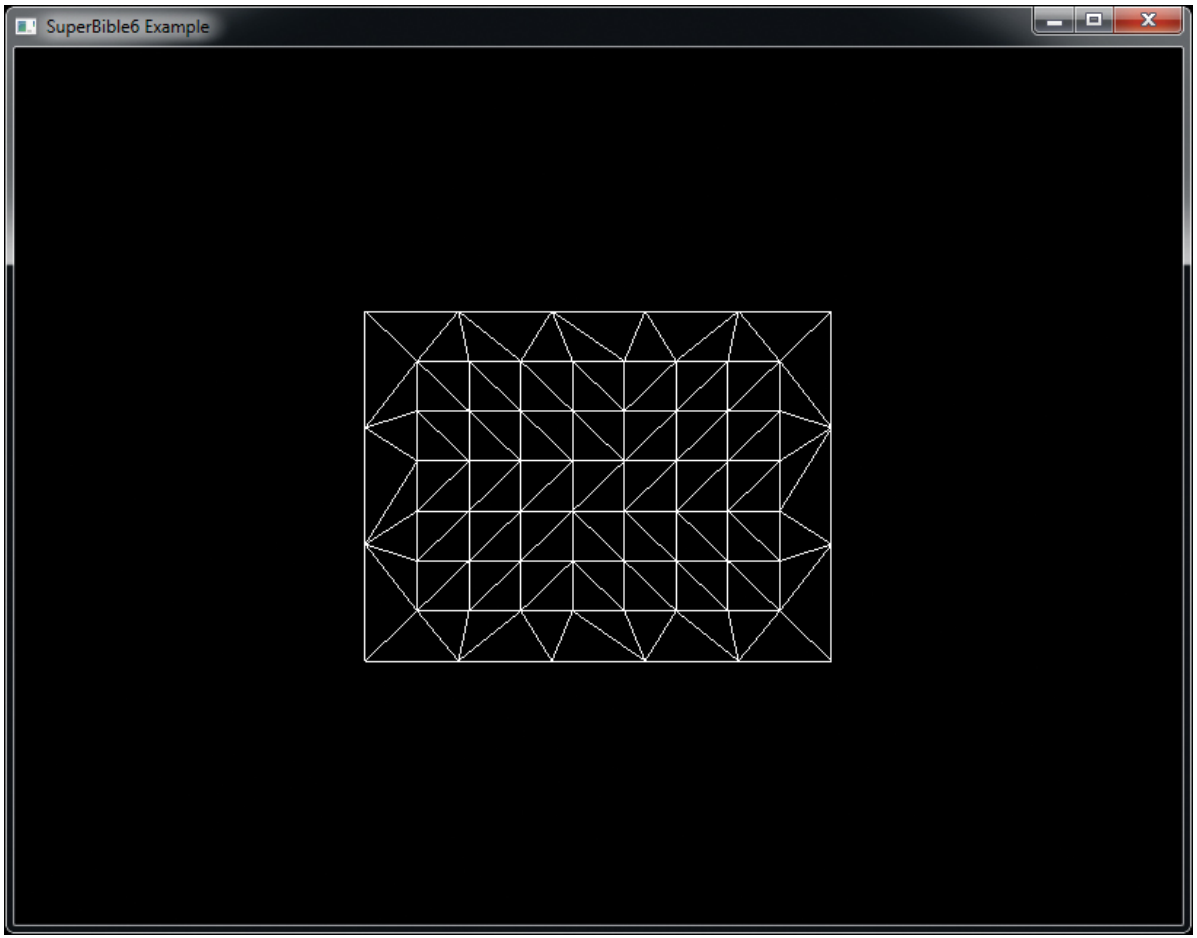
Przykładowy shader kontroli teselacji


```
#version 430 core
layout (vertices = 4) out;

void main(void){
    if (gl_InvocationID == 0){
        gl_TessLevelInner[0] = 9.0;
        gl_TessLevelInner[1] = 7.0;
        gl_TessLevelOuter[0] = 3.0;
        gl_TessLevelOuter[1] = 5.0;
        gl_TessLevelOuter[2] = 3.0;
        gl_TessLevelOuter[3] = 5.0;
    }

    gl_out[gl_InvocationID].gl_Position =
    gl_in[gl_InvocationID].gl_Position;
}
```

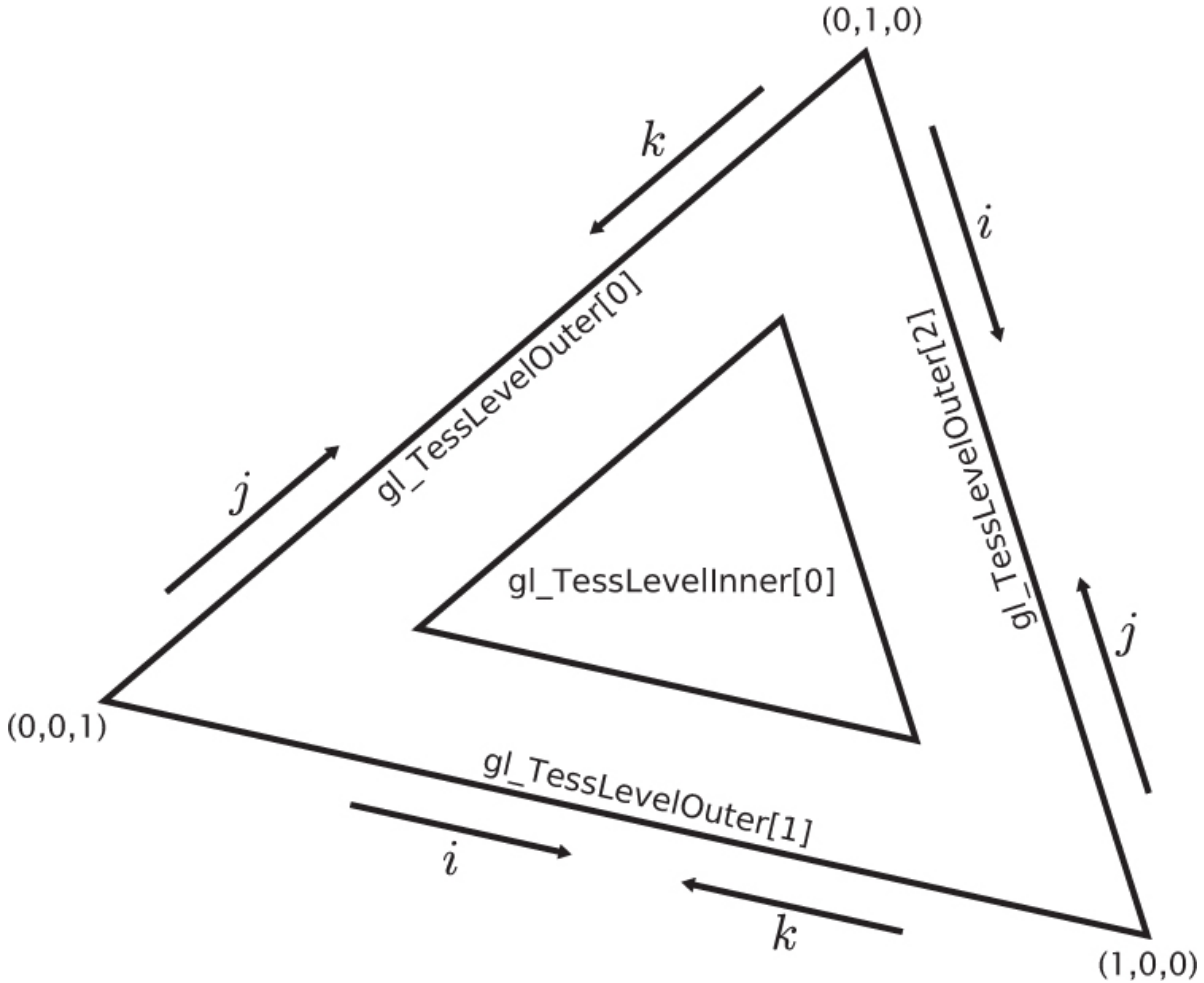
Przykładowy shader ewaluacji teselacji


```
#version 430 core
layout (quads) in;

void main(void){
  // Interpolacja biliniowa
  vec4 p1 = mix(gl_in[0].gl_Position,
                gl_in[1].gl_Position,
                gl_TessCoord.x);
  vec4 p2 = mix(gl_in[2].gl_Position,
                gl_in[3].gl_Position,
                gl_TessCoord.x);
  gl_Position = mix(p1, p2, gl_TessCoord.y);
}
```

Przykładowy wynik



Teselacja w trybie `triangles`



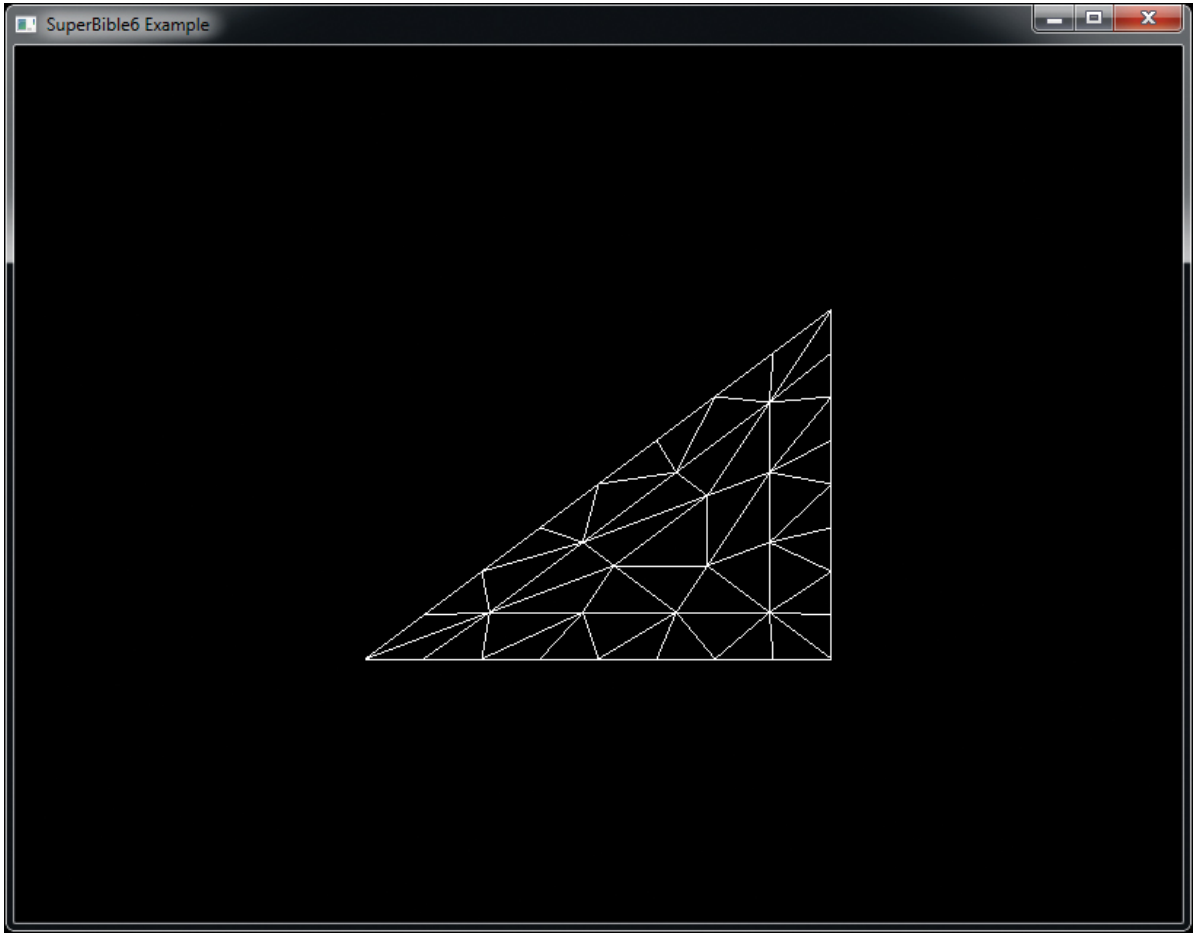
Przykładowy shader kontroli teselacji


```
#version 430 core

layout (vertices = 3) out;

void main(void)
{
    if (gl_InvocationID == 0)
    {
        gl_TessLevelInner[0] = 5.0;
        gl_TessLevelOuter[0] = 8.0;
        gl_TessLevelOuter[1] = 8.0;
        gl_TessLevelOuter[2] = 8.0;
    }

    gl_out[gl_InvocationID].gl_Position =
        gl_in[gl_InvocationID].gl_Position;
}
```

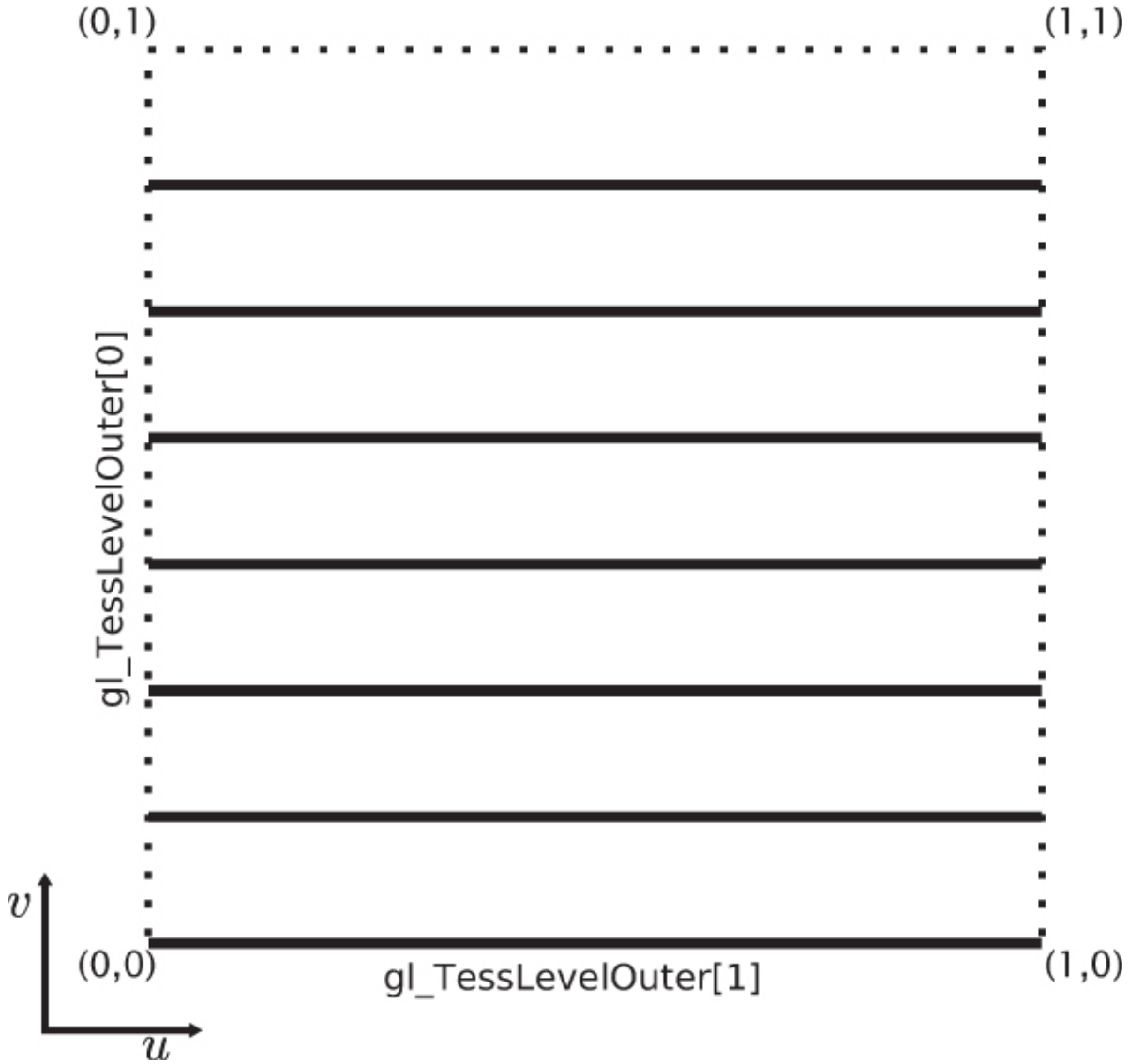
Przykładowy shader ewaluacji teselacji


```
#version 430 core

layout (triangles) in;

void main(void)
{
  gl_Position
    =    (gl_TessCoord.x * gl_in[0].gl_Position)
      + (gl_TessCoord.y * gl_in[1].gl_Position)
      + (gl_TessCoord.z * gl_in[2].gl_Position);
}
```

Przykładowy wynik



Teselacja w trybie `isolines`



Przykładowy shader kontroli teselacji


```
#version 430 core

layout (vertices = 4) out;

void main(void)
{
    if (gl_InvocationID == 0)
    {
        gl_TessLevelOuter[0] = 5.0;
        gl_TessLevelOuter[1] = 5.0;
    }

    gl_out[gl_InvocationID].gl_Position =
        gl_in[gl_InvocationID].gl_Position;
}
```

Przykładowy shader ewaluacji teselacji

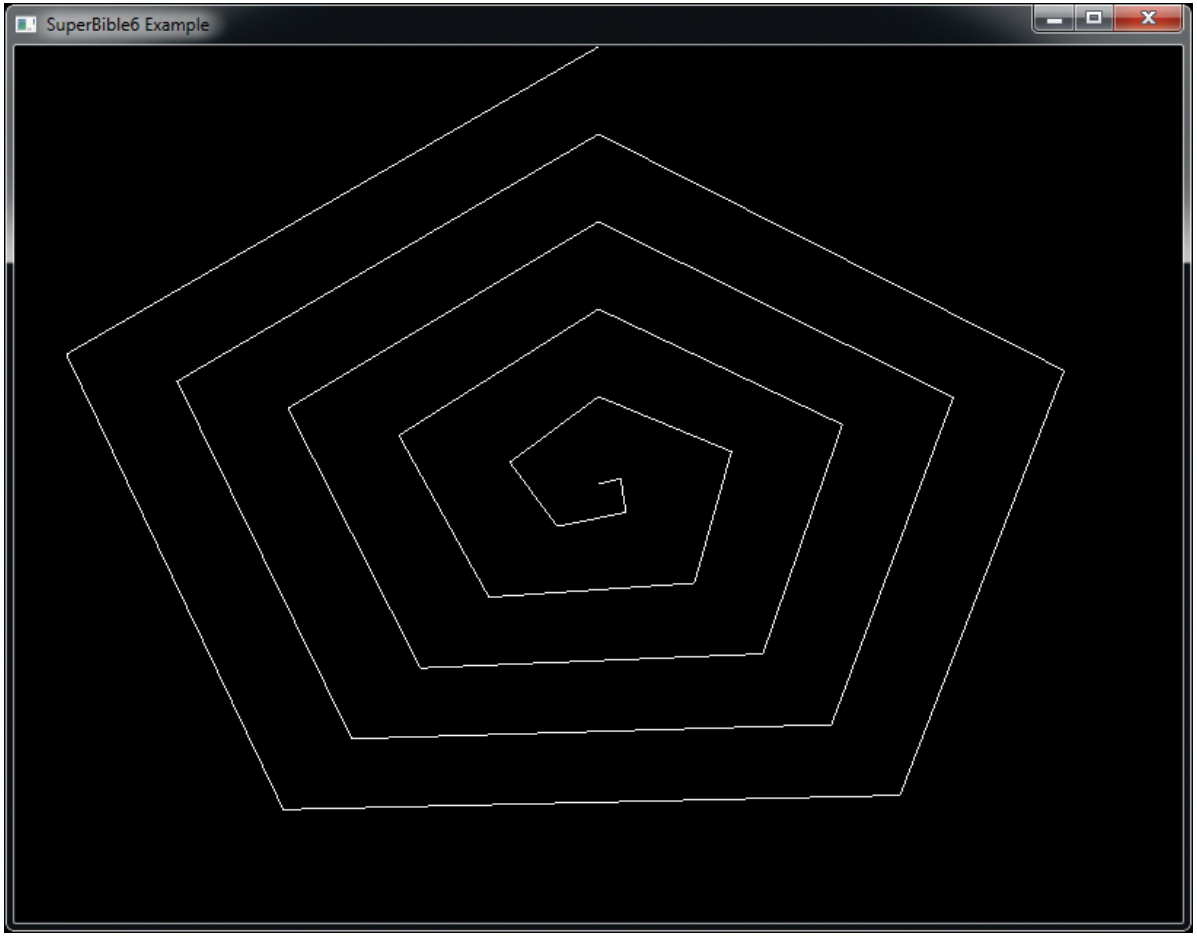

```
#version 430 core

layout (isolines) in;

void main(void)
{
    // Interpolacja bilinowa
    vec4 p1 = mix(gl_in[0].gl_Position,
                  gl_in[1].gl_Position,
                  gl_TessCoord.x);
    vec4 p2 = mix(gl_in[2].gl_Position,
                  gl_in[3].gl_Position,
                  gl_TessCoord.x);
    gl_Position = mix(p1, p2, gl_TessCoord.y);
}
```

Przykładowy wynik

Ciekawszy shader ewaluacji teselacji


```
#version 430 core
layout (isolines) in;

//Spirala
void main(void){
  float r = (gl_TessCoord.y + gl_TessCoord.x
                / gl_TessLevelOuter[0]);
  float t = gl_TessCoord.x * 2.0 * 3.14159;
  gl_Position = vec4(sin(t) * r, cos(t) * r,
                     0.5, 1.0);
}
```

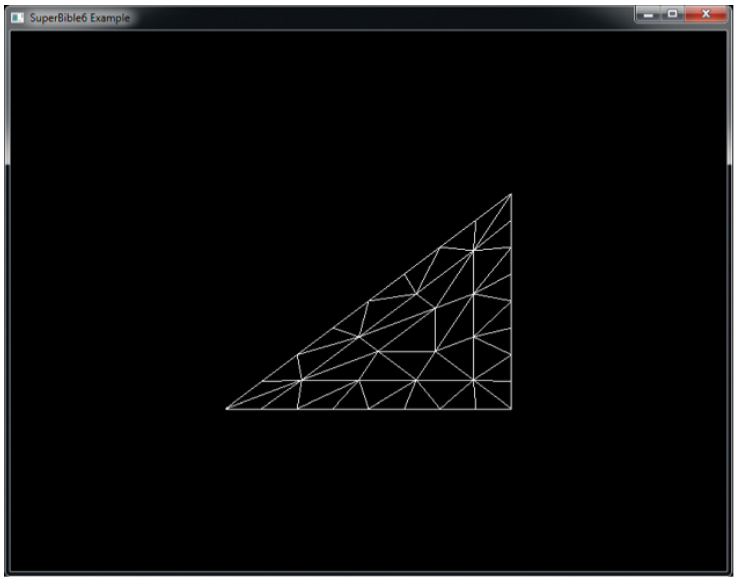
Ciekawszy wynik

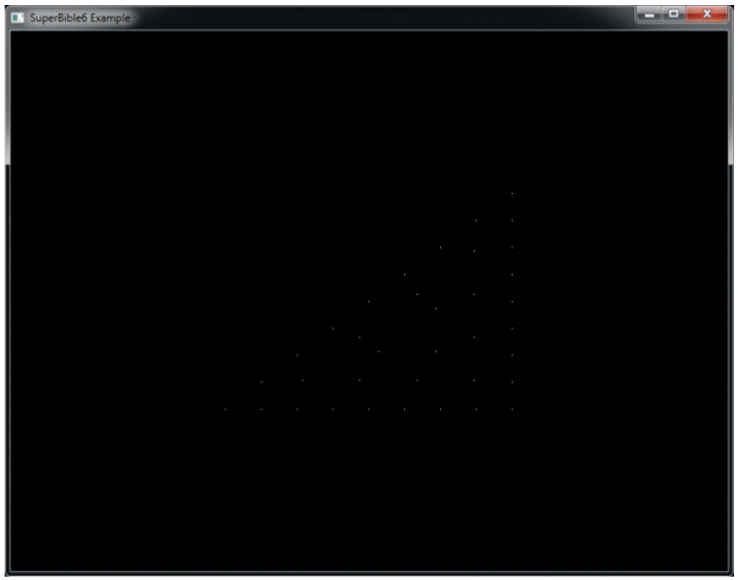


Tryb `point_mode`

- Określa się dodatkowo, razem z `triangles`, `quads` bądź `isolines`
- Przykład:

```
layout (triangles, point_mode) in;
```





Tryby podziału

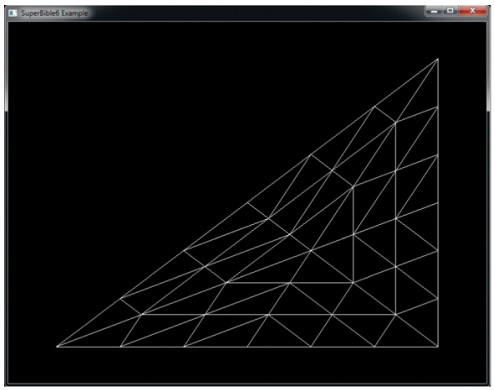
- `layout (equal_spacing) in;`
  - jest domyślny, dzieli na równe odcinki
  - poziom teselacji zaokrągla się w górę do liczby całkowitej
  - powoduje skok przy zmianie poziomu
- `layout (fractional_even_spacing) in;`
  - poziom teselacji zaokrągla się w dół do parzystej liczby całkowitej
  - pozostaje jeden krótki odcinek, który dzieli się na pół
- `layout (fractional_odd_spacing) in;`
  - poziom teselacji zaokrągla się w dół do nieparzystej liczby całkowitej
  - pozostaje jeden krótki odcinek, który dzieli się na pół

Przykład

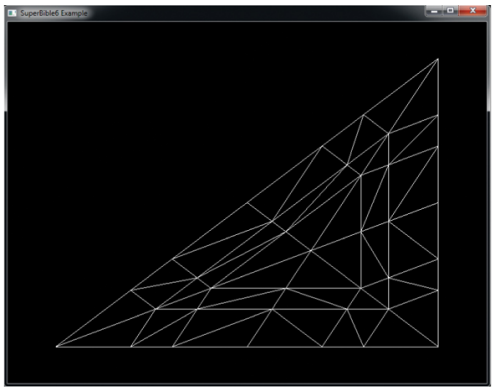
- Poziom teselacji $5{,}3$

```
gl_TessLevelInner[0] = 5.3;
gl_TessLevelOuter[0] = 5.3;
gl_TessLevelOuter[1] = 5.3;
gl_TessLevelOuter[2] = 5.3;
```

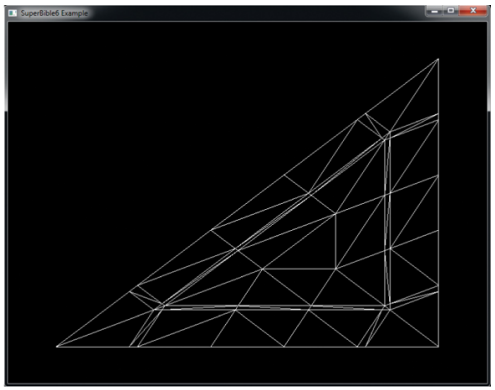
- przy animacji zmiana mesha jest płynna



equal



even



odd

Orientacja trójkątów

- W shaderze ewaluacji teselacji:
  - zgodnie z ruchem wskazówek zegara

```
layout (cw) in;
```

  - przeciwnie do ruchu wskazówek zegara

```
layout (ccw) in;
```

Przekazywanie danych między shaderami

- W shaderze kontroli teselacji:

```
out VS_OUT
{
    vec4        foo;
    vec3        bar;
    int         baz
} vs_out;
```

- W shaderze ewaluacji teselacji:

```
in VS_OUT
{
    vec4        foo;
    vec3        bar;
    int         baz
} tcs_in;
```

- Zmienne `patch`

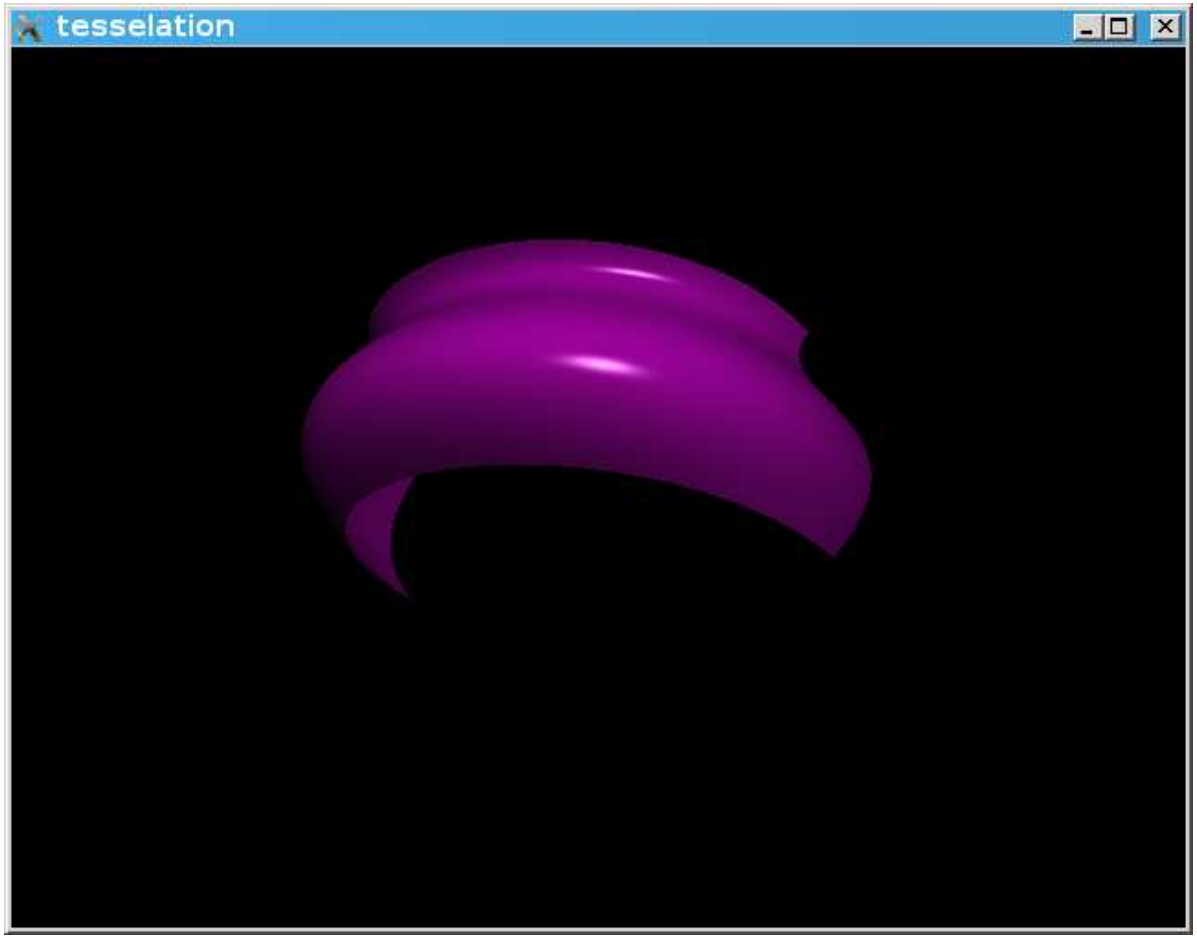
Bez shadera kontroli teselacji

- Poziomy teselacji domyślnie są 1
- Można zmienić w programie:

```
void glPatchParameterfv(
   GLenum pname,
   const GLfloat * values);
```

Płat Béziera

Przykład. Płat Béziera



Shader wierzchołków

- Przeliczamy współrzędne do układu kamery

```
#version 430
layout(location=0) in vec4 in_position;
uniform mat4 model_matrix;
uniform mat4 view_matrix;

void main(void){
    gl_Position = (view_matrix
        * model_matrix) * in_position;
}
```

Shader kontroli teselacji


```
#version 430 core

layout (vertices = 16) out;
void main(void){
  if (gl_InvocationID == 0){
      gl_TessLevelInner[0] = 32.0;
      gl_TessLevelInner[1] = 32.0;
      gl_TessLevelOuter[0] = 32.0;
      gl_TessLevelOuter[1] = 32.0;
      gl_TessLevelOuter[2] = 32.0;
      gl_TessLevelOuter[3] = 32.0;
  }

  gl_out[gl_InvocationID].gl_Position =
    gl_in[gl_InvocationID].gl_Position;
}
```

Shader ewaluacji teselacji. Zmienne uniform


```
#version 430 core
layout (quads, equal_spacing, cw) in;

uniform struct PointLight{
    vec4 position;
    vec4 ambient;
    vec4 diffuse;
    vec4 specular;
    vec3 attenuation;
} light;

uniform mat4 projection_matrix;
uniform mat4 view_matrix;
```

Shader ewaluacji teselacji. Dane wyjściowe


```
out TessOut{
    vec3 normal;
    vec3 light_dir;
    vec3 view_dir;
    float dist;
} vertex;
```

Shader ewaluacji teselacji. Funkcja `Bezier`


```
vec4 Bezier(vec4 p0, vec4 p1,
            vec4 p2, vec4 p3, float t)
{
    vec4 q0, q1, q2, r0, r1;

    q0=mix(p0, p1, t);
    q1=mix(p1, p2, t);
    q2=mix(p2, p3, t);
    r0=mix(q0, q1, t);
    r1=mix(q1, q2, t);
    return mix(r0, r1, t);
}
```

Shader ewaluacji teselacji. Funkcja EvaluatePatch


```
vec4 EvaluatePatch(vec2 at){
  vec4 p[4];
  int i;

  for (i = 0; i < 4; i++){
    p[i] = Bezier(gl_in[i + 0].gl_Position,
           gl_in[i + 4].gl_Position,
           gl_in[i + 8].gl_Position,
           gl_in[i + 12].gl_Position,
           at.y);
  }
  return Bezier(p[0], p[1], p[2], p[3], at.x);
}
```

Shader ewaluacji teselacji. `main`


```
void main(void){
  // Wektory styczne
 vec4 p1 = EvaluatePatch(gl_TessCoord.xy);
 vec4 p3 = EvaluatePatch(gl_TessCoord.xy
           + vec2(0.0, epsilon));
 vec4 p2 = EvaluatePatch(gl_TessCoord.xy
           + vec2(epsilon, 0.0));

 vec3 v1 = normalize(p2.xyz/p2.w - p1.xyz/p1.w);
 vec3 v2 = normalize(p3.xyz/p3.w - p1.xyz/p1.w);

 ..............................
```

Shader ewaluacji teselacji. `main`, cd


```
.....................................
  // Wektor normalny
  vertex.normal = normalize(cross(v1, v2));

  vertex.light_dir
   = normalize(light_position.xyz-p1.xyz/p1.w);
  vertex.view_dir = normalize(-p1.xyz);
  vertex.dist = distance(light_position, p1);

  gl_Position = projection_matrix * p1;
}
```

Shader fragmentów. Światło i materiał


```
#version 430 core
layout (location = 0) out vec4 color;

uniform struct PointLight{
    vec4 position;
    vec4 ambient;
    vec4 diffuse;
    vec4 specular;
    vec3 attenuation;
} light;
uniform struct Material{
  vec4  ambient;
  vec4  diffuse;
  vec4  specular;
  vec4  emission;
  float shininess;
} material;
```

Shader fragmentów. Dane wejściowe i λ

- $\lambda = \pm 1$ służy do zmiany kierunku wektora normalnego

```
uniform float lambda;

in TessOut{
    vec3 normal;
    vec3 light_dir;
    vec3 view_dir;
    float dist;
} vertex;
```

Shader fragmentów. `main` — tłumienie i normalizacja danych wejściowych


```
void main(void){
  float attenuation = 1.0 /(light.attenuation[0]
    light.attenuation[1] * vertex.dist +
    light.attenuation[2] * vertex.dist
        * vertex.dist);

  vec3 normal = normalize(vertex.normal)*lambda;
  vec3 light_dir = normalize(vertex.light_dir);
  vec3 view_dir  = normalize(vertex.view_dir);
```

Shader fragmentów. Obliczenie oświetlenia


```
  color = material.ambient * light.ambient;
  float n_dot_l = max(dot(normal, light_dir),
                      0.0);
  color += material.diffuse * light.diffuse
                      * n_dot_l;
  float r_dot_v_pow =
    max(pow(
     dot(reflect(-light_dir, normal), view_dir),
           material.shininess), 0.0);
  color += material.specular
    * light.specular * r_dot_v_pow;
  color *= attenuation;
  color += material.emission;

}
```

Klasa Bezier


```
class Bezier: public MovableModel,
                    public MaterialModel{
  public:
    void Initialize(const float points [16][4]);
    void Draw(Program & program);
    ~Bezier();
  private:
    GLuint vao_;
    GLuint vertices_;

.............................
```

Konstruktor


```
Bezier::Bezier(){
    velocity_=60;
    rotation_angle_=0;
    model_matrix_.SetUnitMatrix();
    animating_=true;
}
```

Destruktor


```
glBindVertexArray(vao_);
glDisableVertexAttribArray(0);
glBindBuffer(GL_ARRAY_BUFFER, 0);
glDeleteBuffers(1, &vertices_);
glBindVertexArray(0);
glDeleteVertexArrays(1, &vao_);
```

Inicjalizacja


```
glGenVertexArrays(1, &vao_);
glBindVertexArray(vao_);

glGenBuffers(1, &vertices_);
glBindBuffer(GL_ARRAY_BUFFER, vertices_);
glBufferData(GL_ARRAY_BUFFER,
   64*sizeof(float), points, GL_STATIC_DRAW);
glVertexAttribPointer(0, 4, GL_FLOAT,
   GL_FALSE, 0, nullptr);
glEnableVertexAttribArray(0);

glBindVertexArray(0);
```

Wyświetlenie. Pierwszy bok


```
glBindVertexArray(vao_);
glUseProgram(program);

program.SetModelMatrix(model_matrix_);
program.SetMaterial(material_);

glEnable(GL_CULL_FACE);
glCullFace(GL_BACK);

program.SetLambda(1);
glFrontFace(GL_CW);
glPatchParameteri(GL_PATCH_VERTICES, 16);
glDrawArrays(GL_PATCHES, 0, 16);
```

Wyświetlenie. Drugi bok


```
program.SetLambda(-1);
glFrontFace(GL_CCW);
glPatchParameteri(GL_PATCH_VERTICES, 16);
glDrawArrays(GL_PATCHES, 0, 16);

glDisable(GL_CULL_FACE);
glBindVertexArray(0);
glUseProgram(0);
```

Klasa Program


```
Initialize(const char * vertex,
           const char* fragment,
           const char * tessControl,
           const char *tessEval);
private:
    GLuint program_;

    GLuint vertex_shader_;
    GLuint fragment_shader_;
    GLuint tess_control_shader_;
    GLuint tess_eval_shader_;
    GLuint lambda_location_;

 ..............................
```

Klasa `Window`. Punkty kontrolne


```
const GLfloat kBezierOne[16][4] ={
 {-2, 1, 0, 1},
 {-2.0f/3.0f, 1.0f/3.0f, 4.0f/3.0f, 1.0f/3.0f},
 { 2.0f/3.0f, 1.0f/3.0f, 4.0f/3.0f, 1.0f/3.0f},
 {2, 1, 0, 1},
  {-1.5, 0.5, 0,1},
  {-0.5, 1.0f/6.0f, 1, 1.0f/3.0f},
  { 0.5, 1.0f/6.0f, 1, 1.0f/3.0f},
  {1.5, 0.5, 0, 1},
    {-3, 0, 0,1}, {-1, 0, 2, 1.0f/3.0f},
    { 1, 0, 2, 1.0f/3.0f}, {3, 0, 0, 1},
 {-2, -1, 0, 1},
 {-2.0f/3.0f, -1.0f/3.0f, 4.0f/3.0f, 1.0f/3.0f},
 { 2.0f/3.0f, -1.0f/3.0f, 4.0f/3.0f, 1.0f/3.0f},
 {2, -1, 0, 1}
};
```