Programowanie 3W grafiki w OpenGL. Teselacja. Płaty Béziera

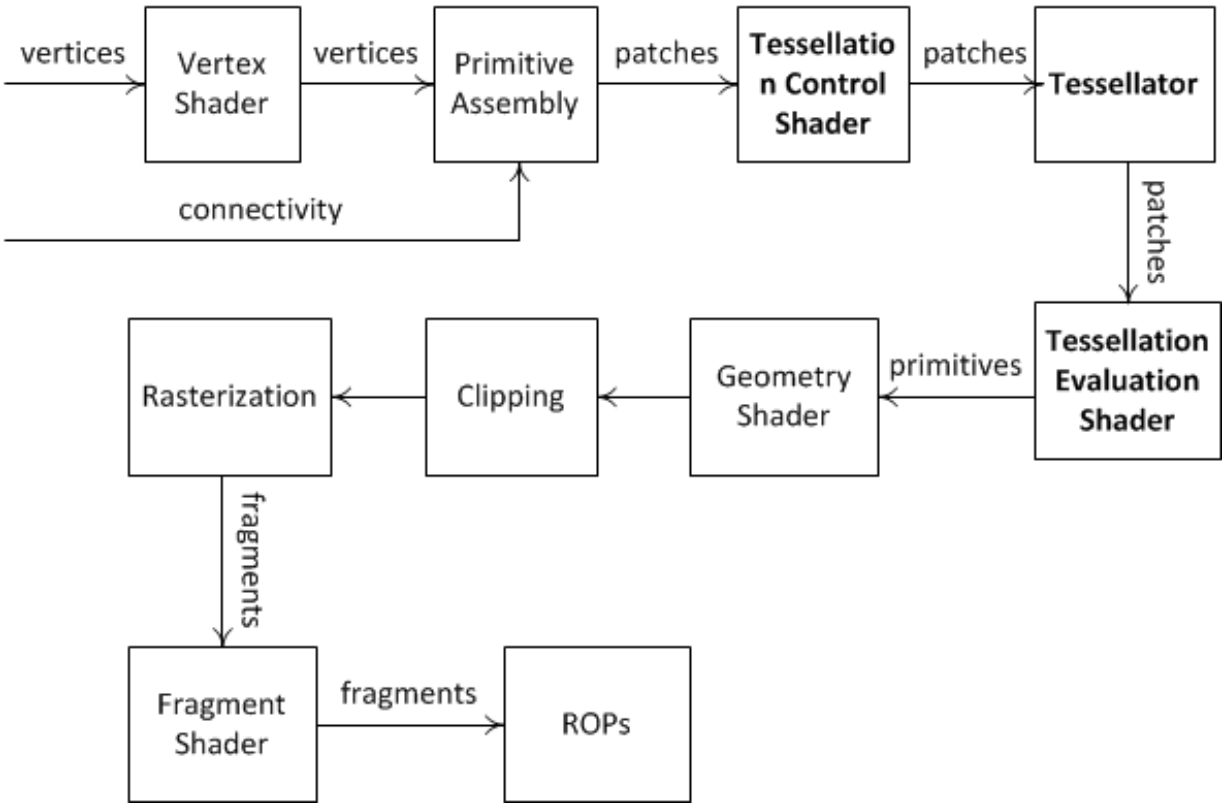
Aleksander Denisiuk
Polsko-Japońska Akademia Technik Komputerowych
Wydział Informatyki w Gdańsku
ul. Brzegi 55
80-045 Gdańsk

denisjuk@pja.edu.pl

Najnowsza wersja tego dokumentu dostępna jest pod adresem
http://users.pja.edu.pl/~denisjuk/

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

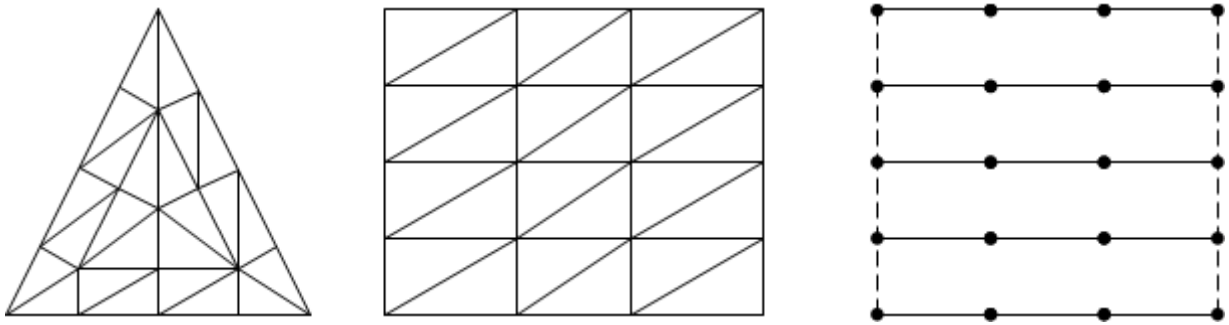
- `gl_out[]` — dane wierzchołków
  - `gl_Position`
  - `gl_PointSize`
  - `gl_ClipDistance[]`
- `gl_TessLevelOuter[]` — rozbicie granic prymitywu
- `gl_TessLevelInner[]` — rozbicie wnętrza prymitywu
- inne dane wyjściowe, opisane jako tabele, dla każdego wierzchołka
- zmienne typy `patch`

```
patch out float my_value
```

Funkcja `barrier()`

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

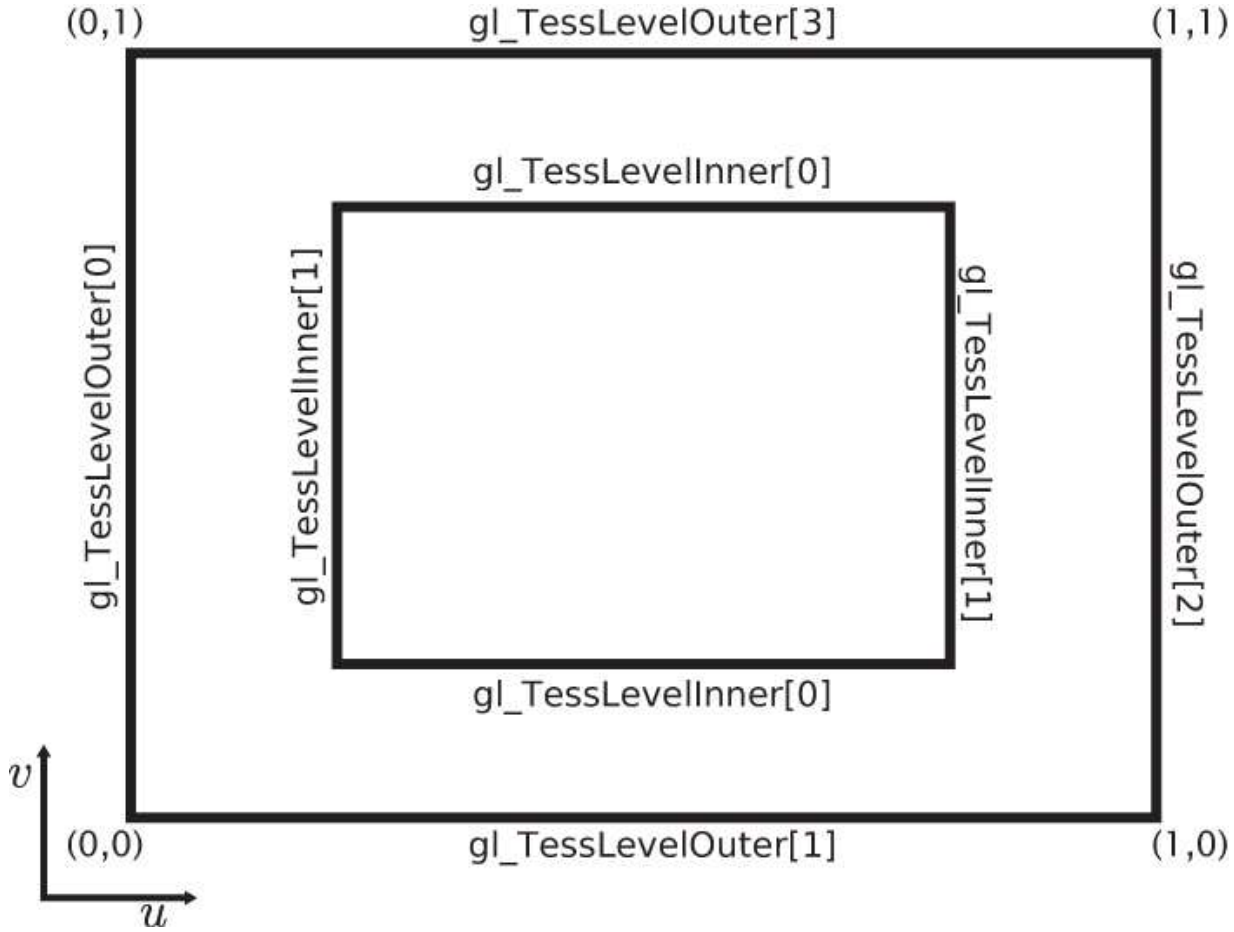
Shader ewaluacji teselacji



- 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



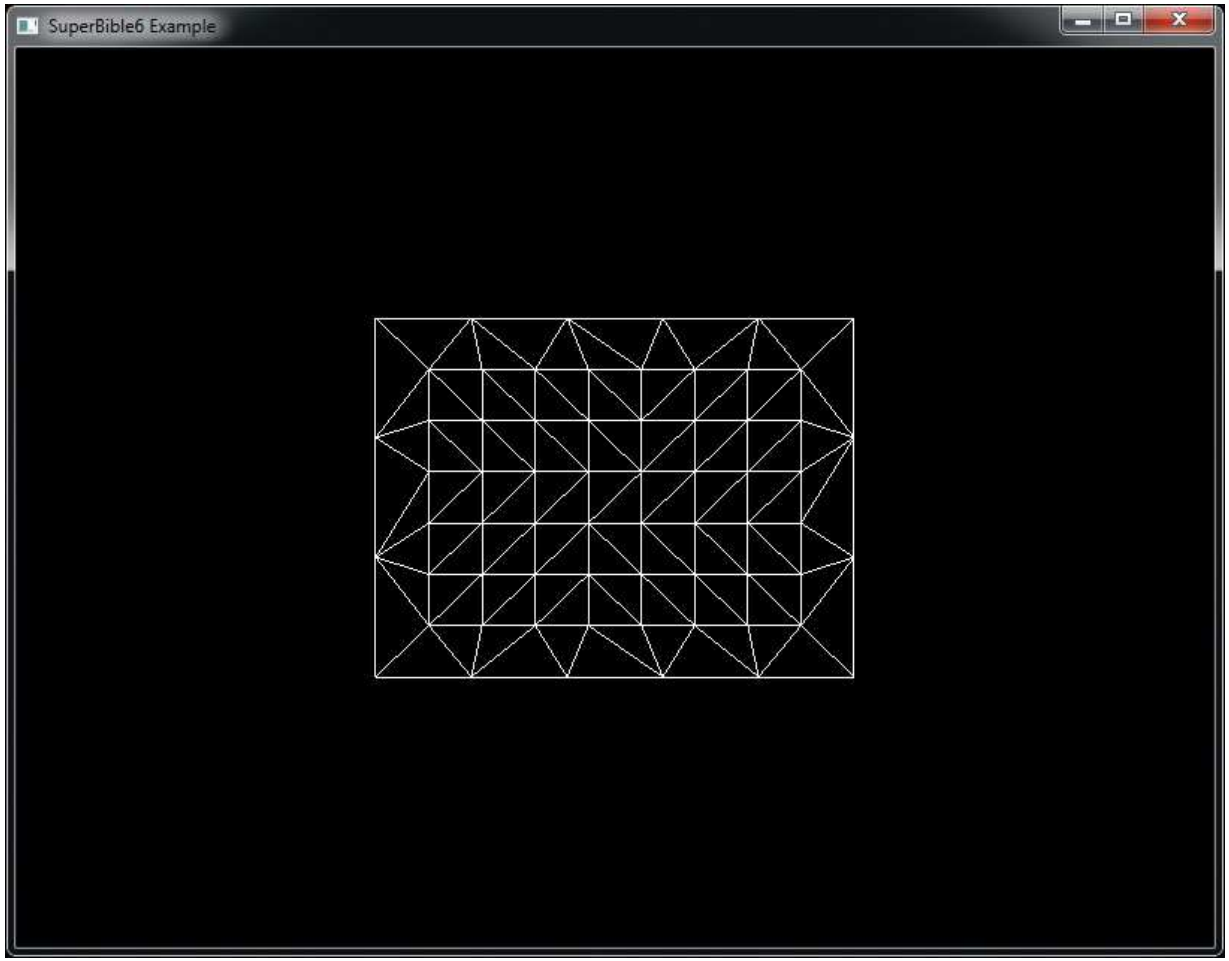
Przykładowy shader kontroli teselacji


```
#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;
}
```

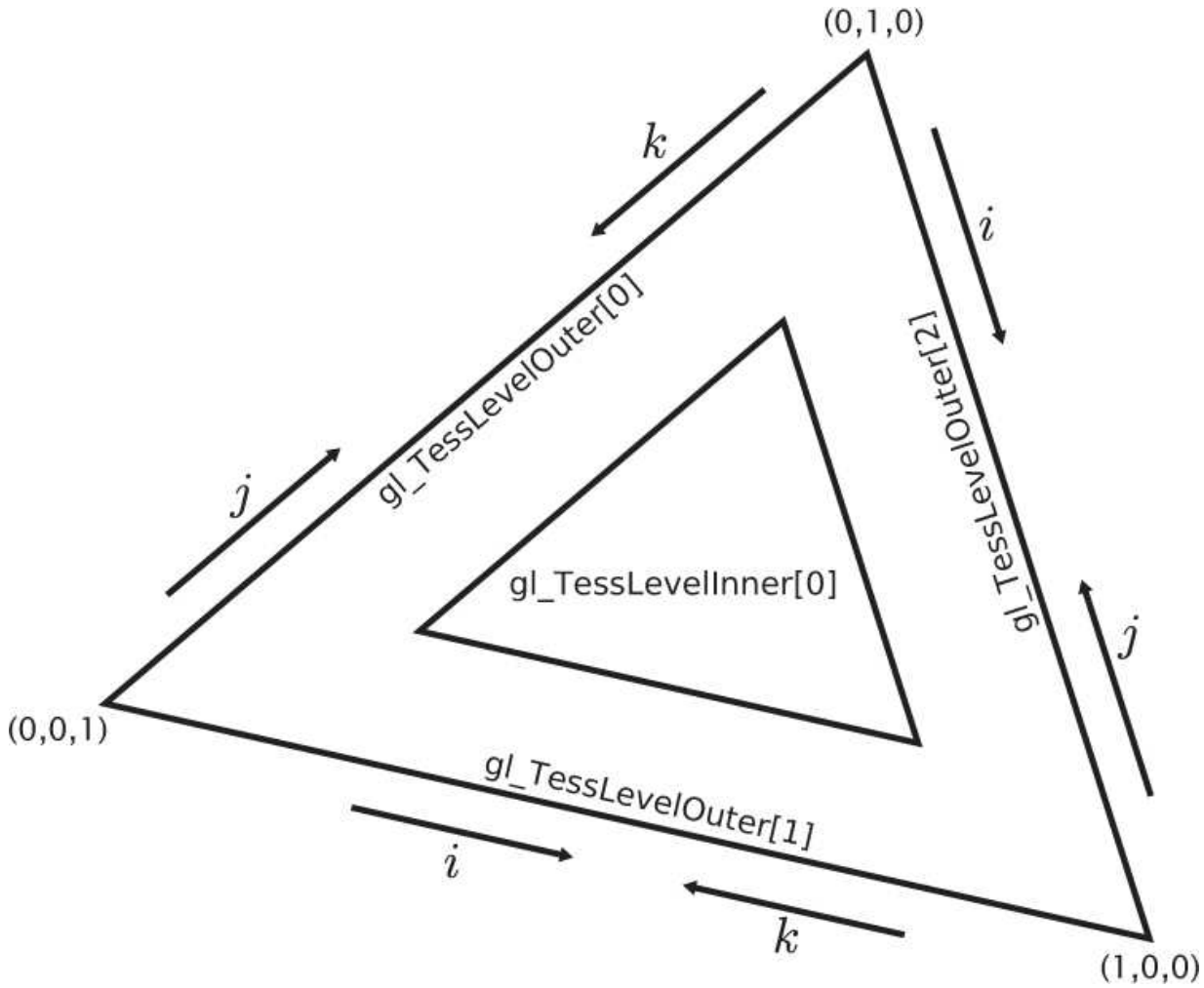
Przykładowy shader ewaluacji teselacji


```
#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`



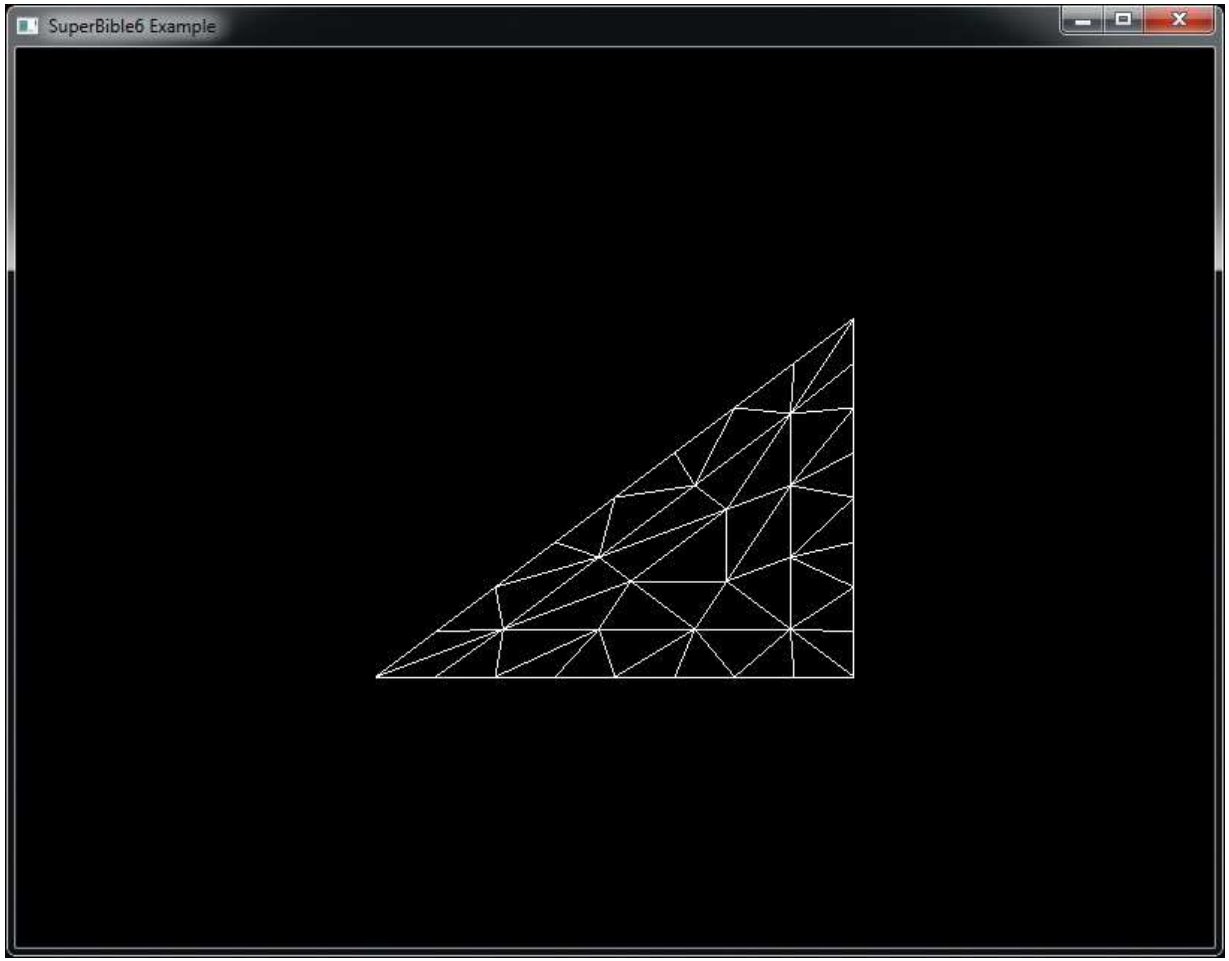
Przykładowy shader kontroli teselacji


```
#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;
}
```

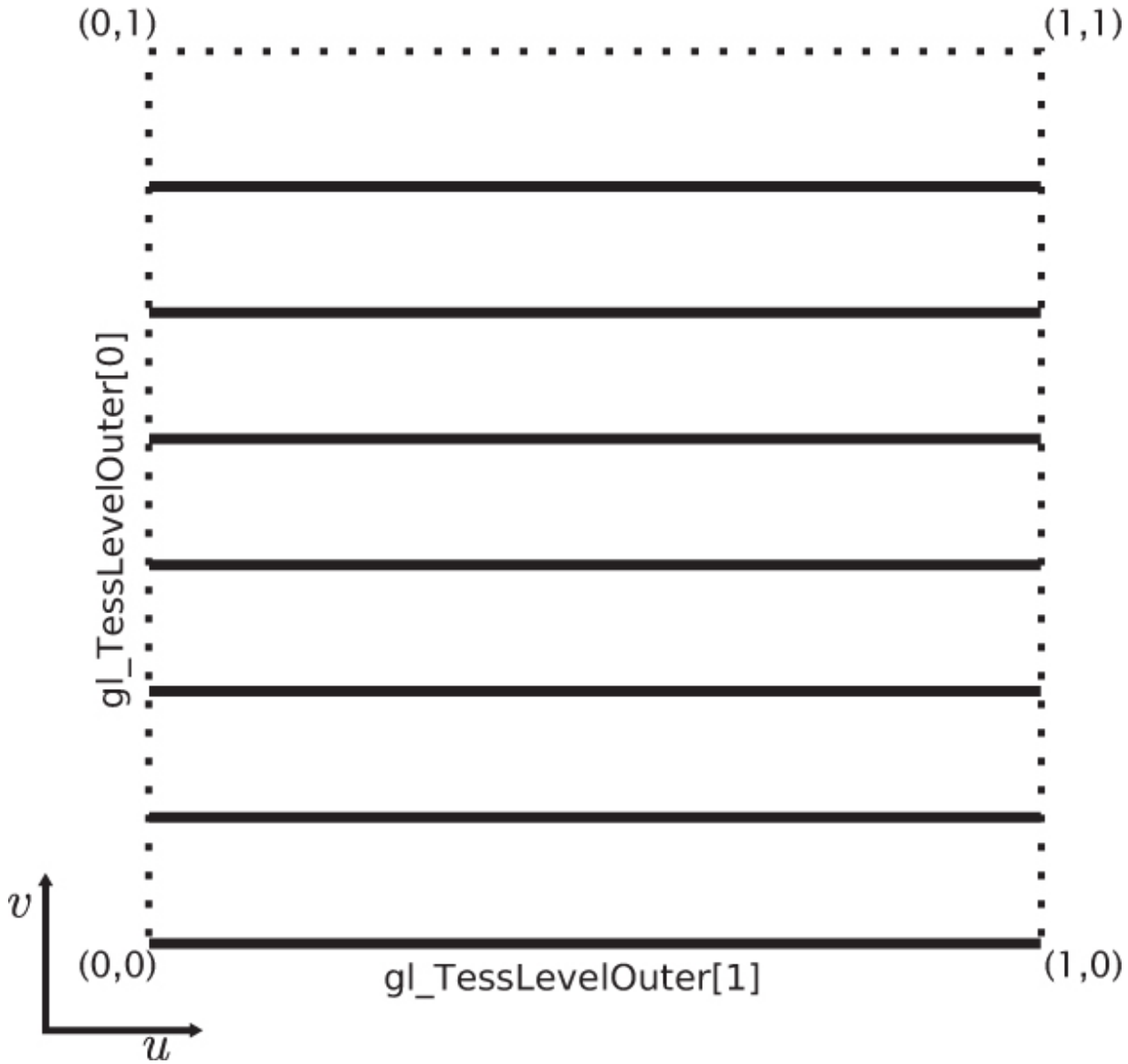
Przykładowy shader ewaluacji teselacji


```
#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

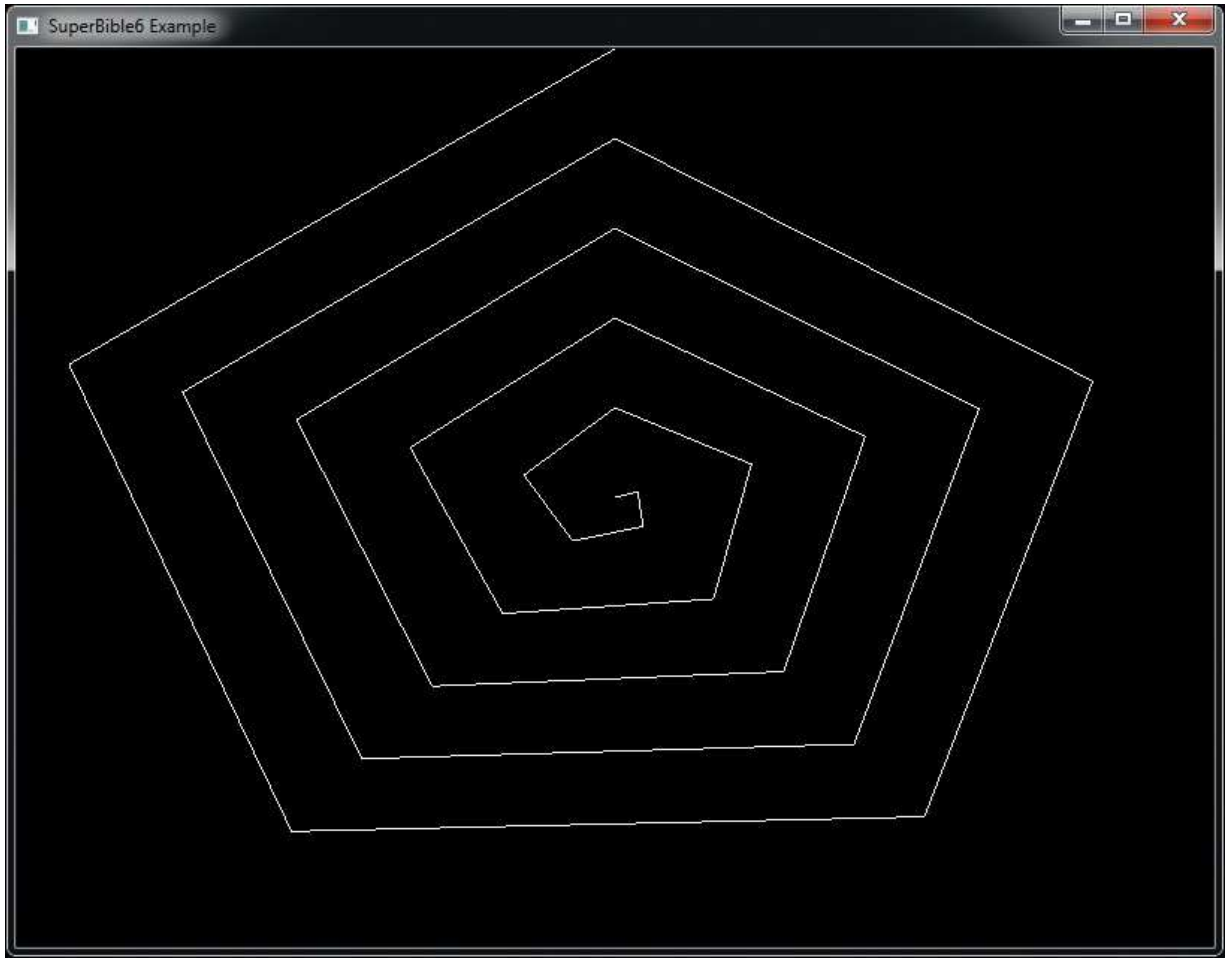

```
#version 430 core

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);
}
```

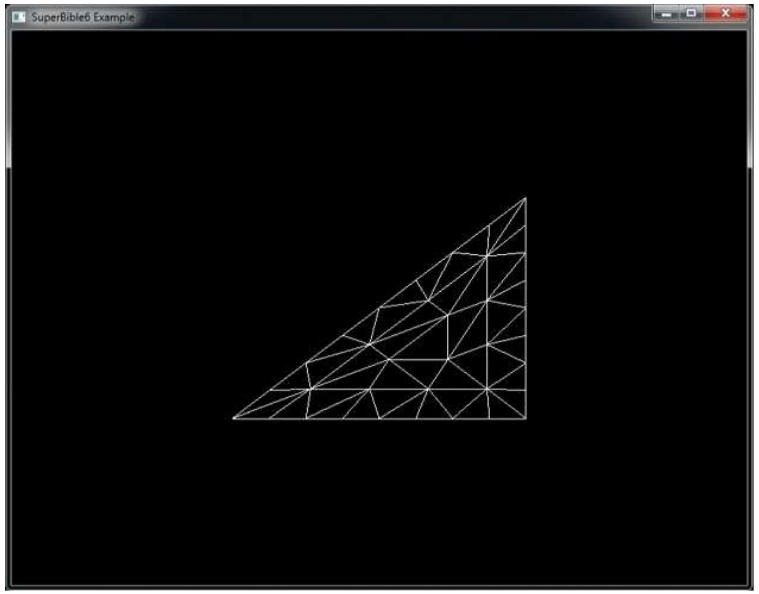
Ciekawszy wynik

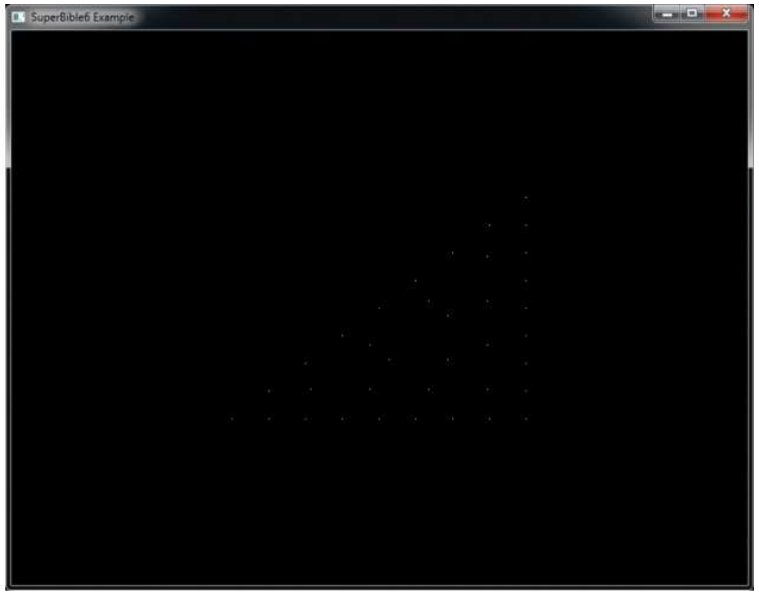


Tryb `point_mode`

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

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





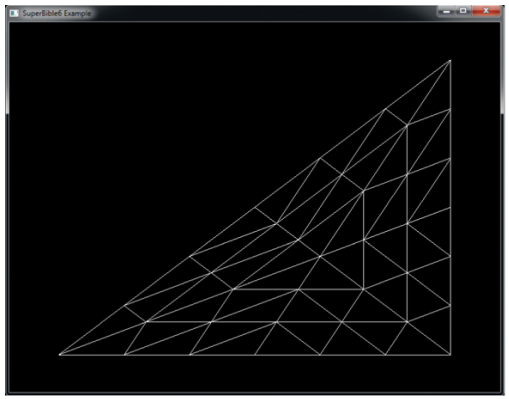
Tryby podziału

- `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

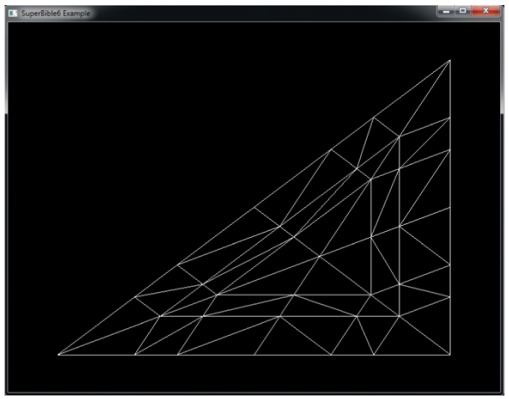
- Poziom teselacji 5,3

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

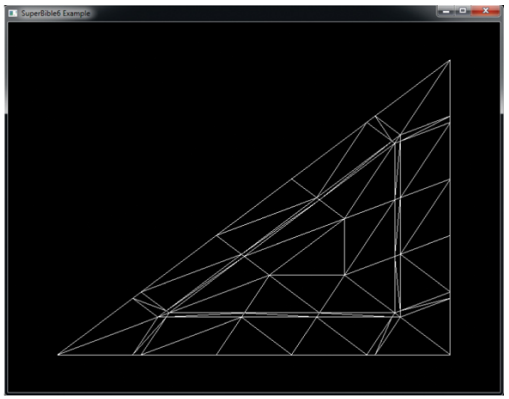
- przy animacji zmiana mesha jest płynna



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;
```

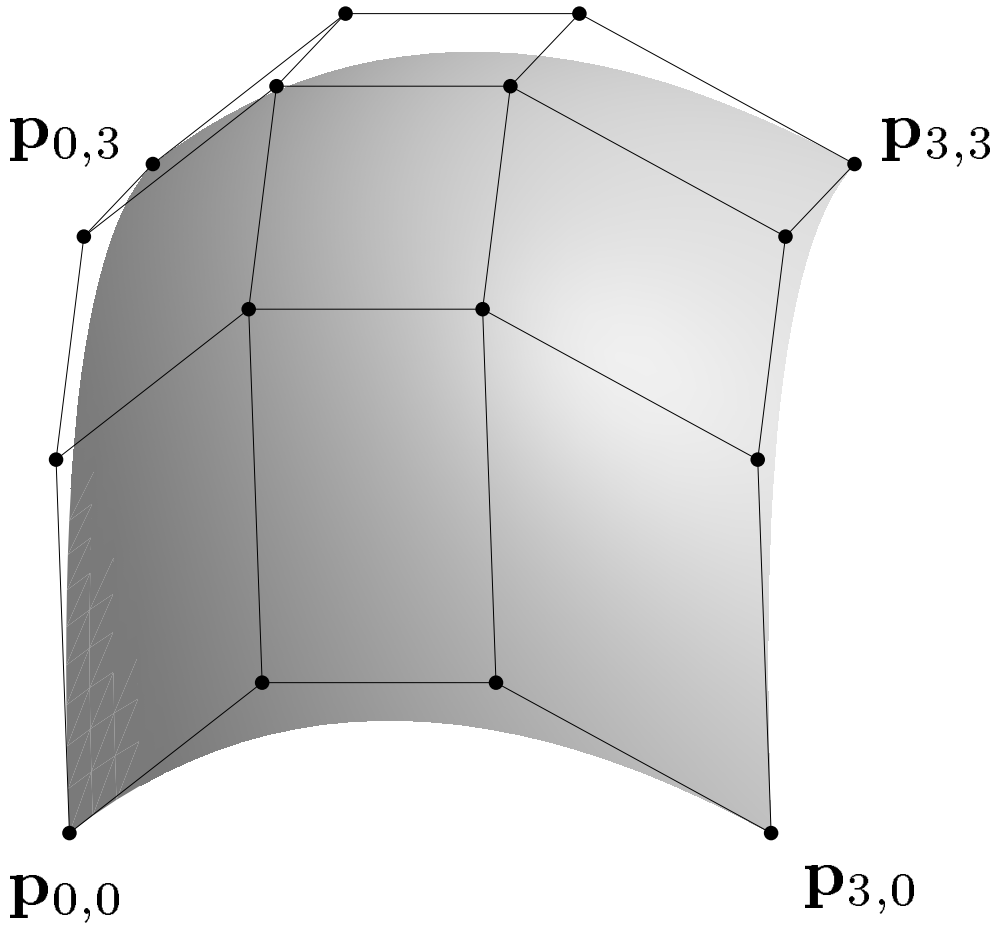
- Zmienne `patch`

Bez shadera kontroli teselacji

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

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

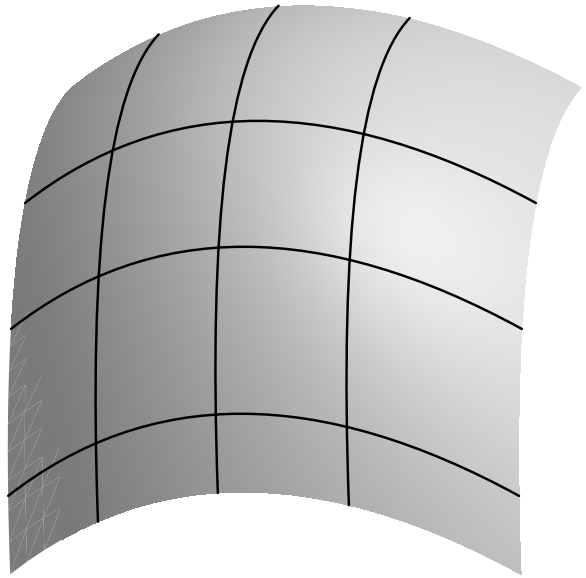
Płaty Béziera



Powierzchnie Béziera trzeciego stopnia

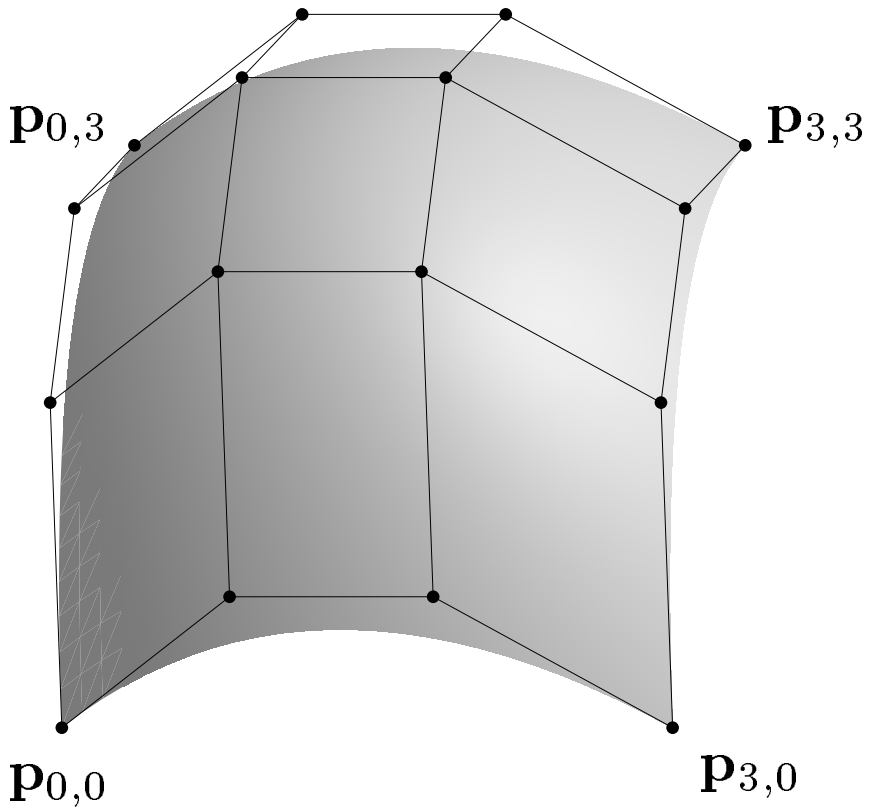
$$q(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} B_i(u)B_j(v)p_{i,j} =$$
$$= \sum_{i=0}^{3}\left(B_i(u)\sum_{j=0}^{3} B_j(v)p_{i,j}\right) =$$
$$= \sum_{j=0}^{3} B_j(v)\left(\sum_{i=0}^{3} B_i(u)p_{i,j}\right),$$
$$(u,v) \in [0,1]\times[0,1]$$

Przekrój powierzchni Béziera



- $q(u,v) = \sum_{i=0}^{3} \left(B_i(u) \sum_{j=0}^{3} B_j(v) p_{i,j} \right)$
- $r_i = \sum_{j=0}^{3} B_j(v) p_{i,j}, \quad s_j = \sum_{i=0}^{3} B_i(u) p_{i,j}$

Graniczne linie powierzchni Béziera



- $v = 0, \quad u \in [0, 1]$: granica „przednia”, $p_{i,0}$
- $u = 0, \quad v \in [0, 1]$: granica „lewa”, $p_{0,j}$

Pochodne cząstkowe powierzchni Béziera

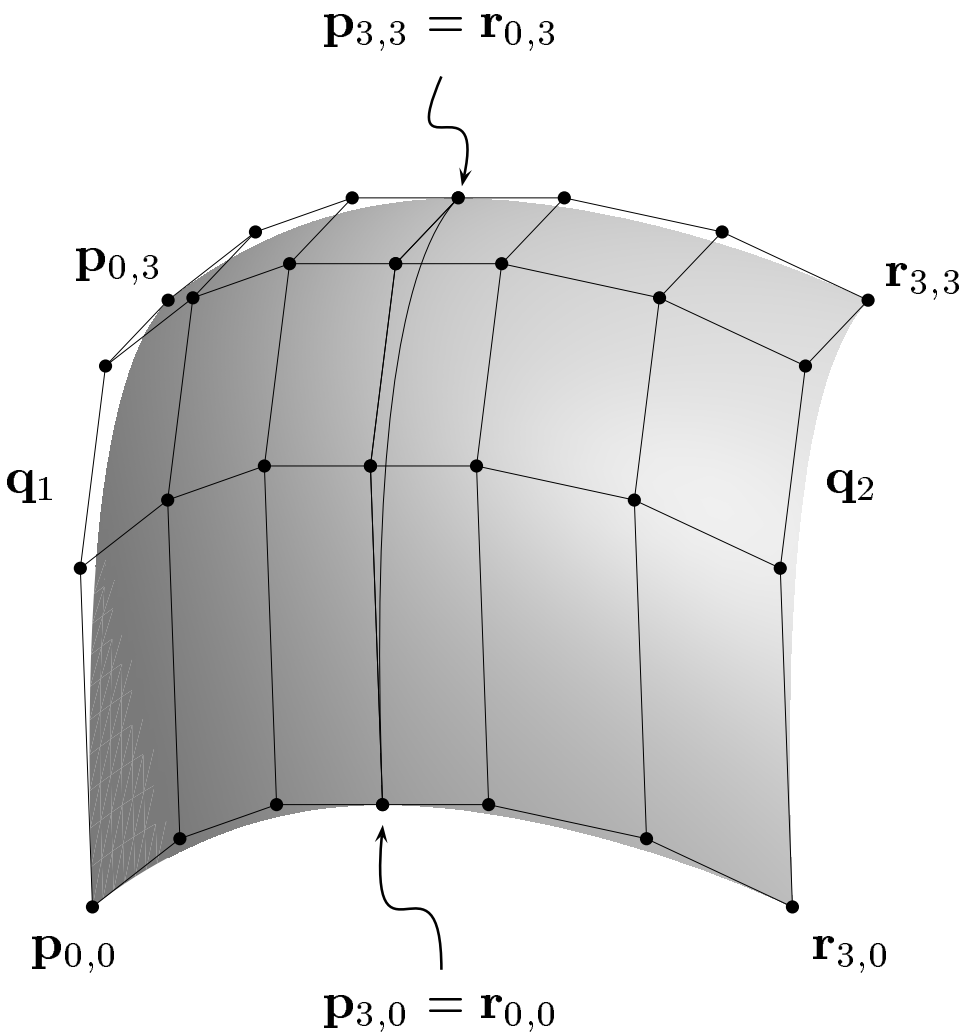
$$\frac{\partial q}{\partial v}(u,0) = \sum_{i=0}^{3} 3B_i(u)(p_{i,1} - p_{i,0})$$

$$\frac{\partial q}{\partial v}(u,1) = \sum_{i=0}^{3} 3B_i(u)(p_{i,3} - p_{i,2})$$

$$\frac{\partial q}{\partial u}(0,v) = \sum_{i=0}^{3} 3B_j(v)(p_{1,j} - p_{0,j})$$

$$\frac{\partial q}{\partial v}(1,v) = \sum_{i=0}^{3} 3B_j(v)(p_{3,j} - p_{3,j})$$

Sklejane powierzchnie Béziera



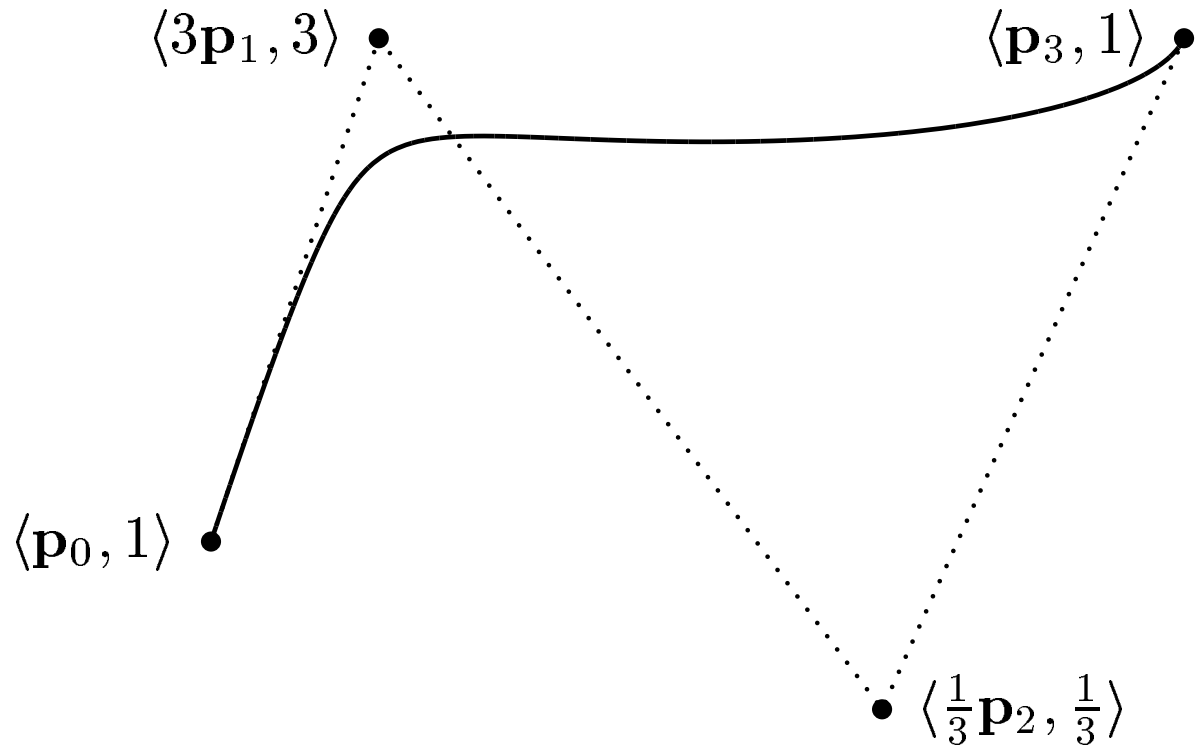
$p_i = (x : y : z : w)$,

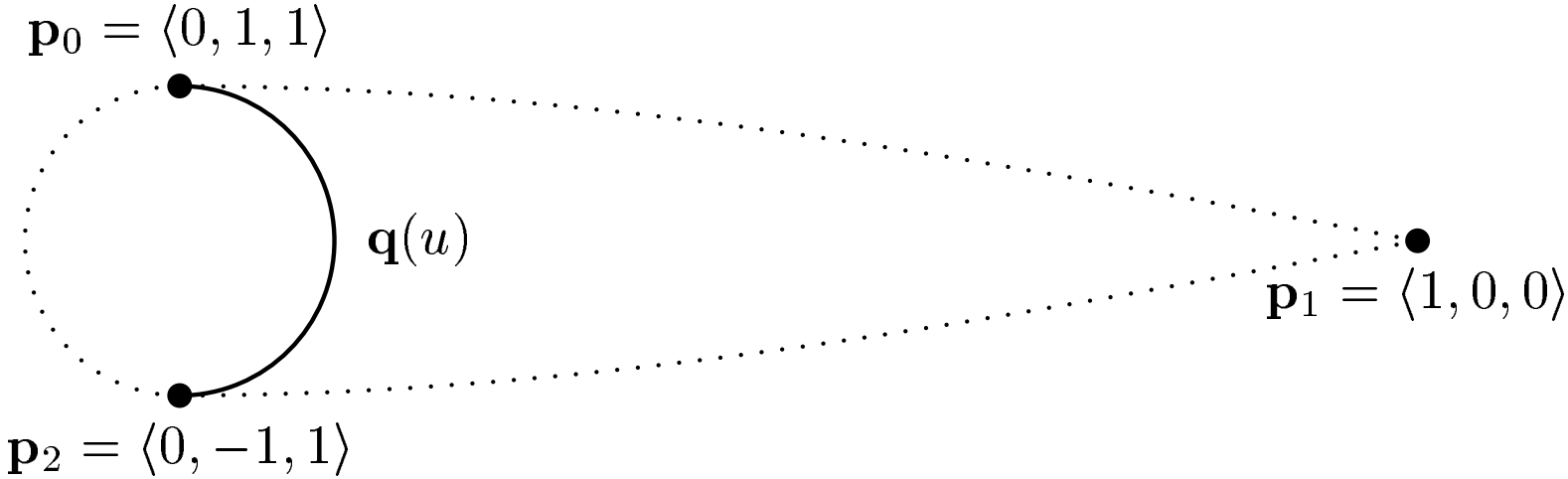
$$q(u) = \sum_i B_i^k(u) p_i$$

- współrzędna w pozwala na powiększenie wagi punktu kontrolnego
- modelowanie krzywych stożkowych
- rzut perspektywiczny krzywej wymiernej jest zawsze krzywą wymierną
- punkty kontrolne mogą być umieszczone w nieskończoności

Powiększenie wagi punktu kontrolnego

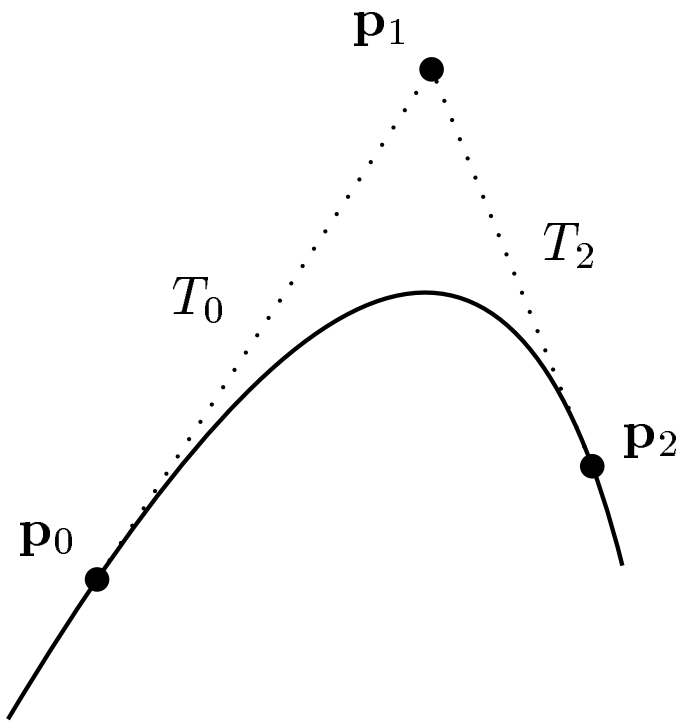
$$q(u) = \sum_i B_i^k(u)(w_i p_i : w_i) \sim \sum_i \frac{w_i B_i^k(u)}{\sum_j w_j B_i^k(u)} p_i$$

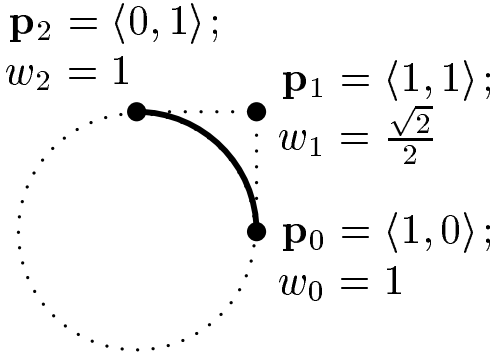


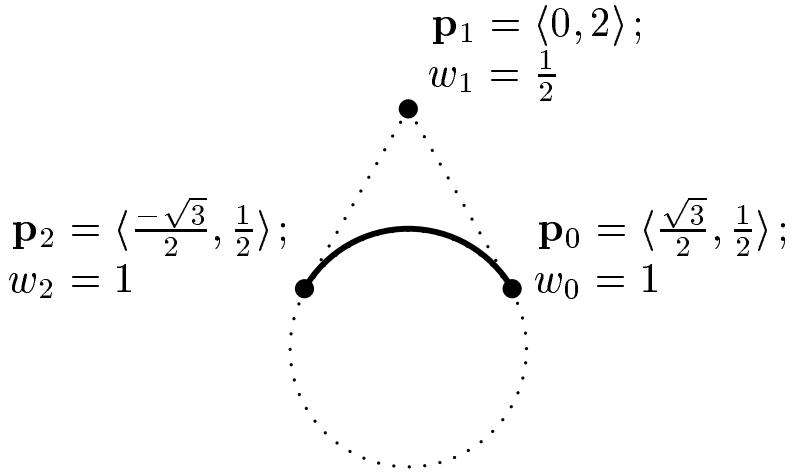


$$
\begin{aligned}
q(u) &= (1-u)^2 p_0 + 2u(1-u)p_1 + u^2 p_2 = \\
&= (2u(1-u) : (1-u)^2 - u^2 : (1-u)^2 + u^2) \sim \\
&\sim \left(\frac{2u(1-u)}{(1-u)^2 + u^2}, \frac{(1-u)^2 - u^2}{(1-u)^2 + u^2} \right)
\end{aligned}
$$

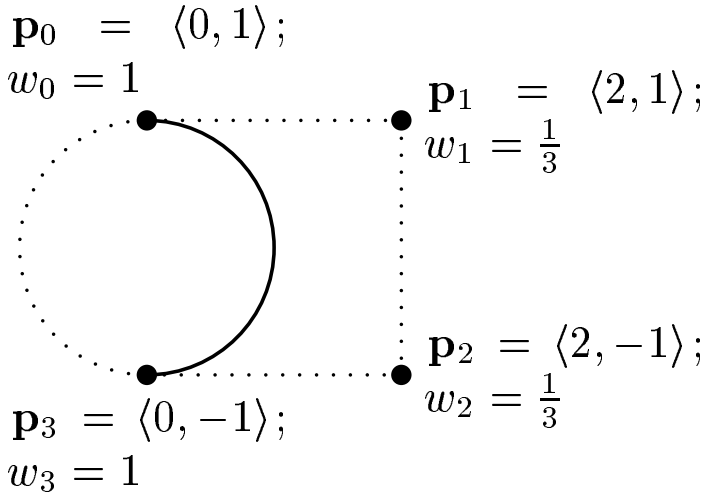
Twierdzenie 1. Niech T_0 i T_2 będą stycznymi do krzywej stożkowej C w punktach p_0 i p_2, p_1 będzie punktem przecięcia T_0 i T_2. Wtedy istnieje waga $w \geqslant 0$ taka, że wymierna krzywa Béziera o punktach kontrolnych $(p_0 : 1)$, $(p_1 : w)$, $(p_2 : 1)$ generuje odcinek krzywej C pomiędzy p_0 a p_2.





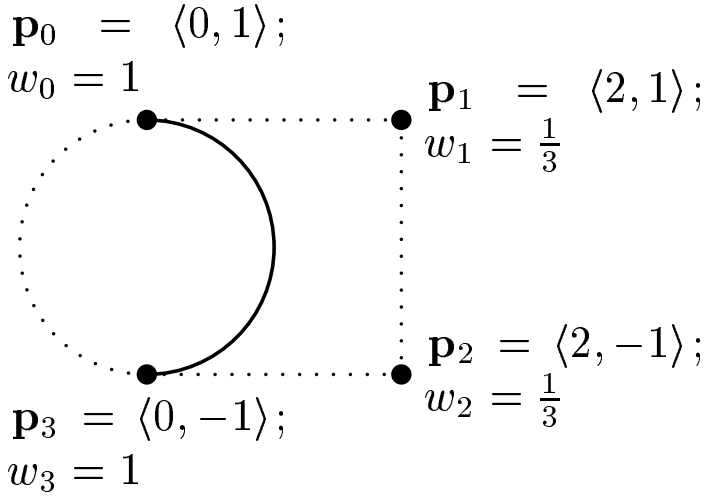


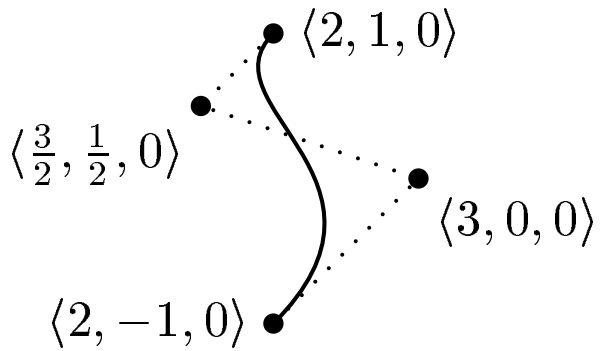
Półokrąg jako krzywa trzeciego stopnia



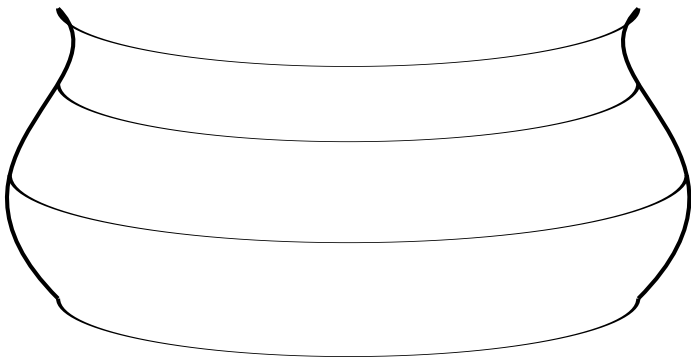
Okrąg o promieniu 2

$$(p_0, p_1, p_2) \mapsto (p_0^* = Mp_0, p_1^* = Mp_1, p_2^* = Mp_2, p_3^* = Mp_3)$$





(a)

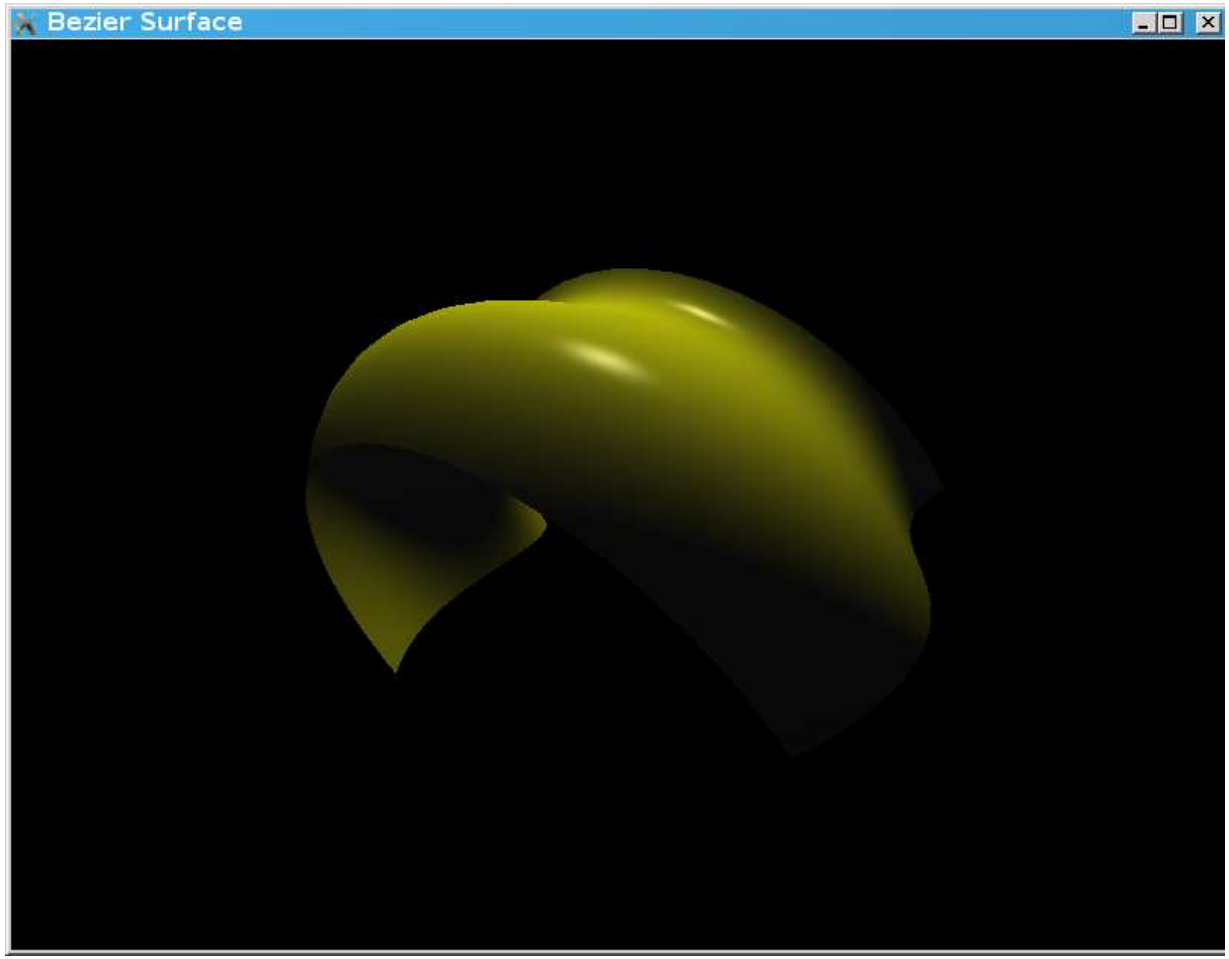


(b)

$$
\begin{array}{cccc}
(-2:1:0:1) & (-\frac{2}{3}:\frac{1}{3}:\frac{4}{3}:\frac{1}{3}) & (-\frac{2}{3}:\frac{1}{3}:\frac{4}{3}:\frac{1}{3}) & (2:1:0:1) \\
(-\frac{3}{2}:\frac{1}{2}:0:1) & (-\frac{1}{2}:\frac{1}{6}:1:\frac{1}{3}) & (\frac{1}{2}:\frac{1}{6}:1:\frac{1}{3}) & (\frac{3}{2}:\frac{1}{2}:0:1) \\
(-3:0:0:1) & (-1:0:2:\frac{1}{3}) & (1:0:2:\frac{1}{3}) & (3:0:0:1) \\
(-2:-1:0:1) & (-\frac{2}{3}:-\frac{1}{3}:\frac{4}{3}:\frac{1}{3}) & (\frac{2}{3}:-\frac{1}{3}:\frac{4}{3}:\frac{1}{3}) & (2:-1:0:1)
\end{array}
$$

Aplikacja

Przykład. Płat Béziera



- Uwaga: płat ma dwie strony, aby poprawnie obliczyć oświetlenie, trzeba każdą rednerować oddzielnie

```
#version 430 core
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. Dane


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

Shader ewaluacji teselacji. Procedura `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. Obliczenie punktu na płacie Béziera


```
vec4 evaluate_patch(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


```
const float epsilon = 0.001;
void main(void){
  vec4 p1 = evaluate_patch(gl_TessCoord.xy);
  vec4 p3 = evaluate_patch(gl_TessCoord.xy
            + vec2(0.0, epsilon));
  vec4 p2 = evaluate_patch(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);
  vec4 light_position = view_matrix * light.position;
  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;
}
```

```
#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.emission;
    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;
}
```

- `Window` — odpowiada za kontekst
- `Bezier` — model: VAO, renderowanie
- `Program` — program: kompilacja shaderów
- `matma.h` — implementacja algebry macierzy
- `material.h` — struktura dla materiału
- `light.h` — struktura dla światła
- `glerror.h` — definicje, związane z debugowaniem

Struktura dla światła


```
typedef struct PointLight{
  GLfloat position[4];
  GLfloat ambient[4];
  GLfloat diffuse[4];
  GLfloat specular[4];
  GLfloat attenuation[3];
} PointLight;
```

Struktura dla materiału


```
typedef struct Material{
  GLfloat ambient[4];
  GLfloat diffuse[4];
  GLfloat specular[4];
  GLfloat emission[4];
  GLfloat shininess;
} Material;
```

Klasa Program. Dane publiczne


```
class Program{
public:
    void Initialize(const char* vertex_shader_file,
                    const char* fragment_shader_file,
                    const char* tess_control_shader_file,
                    const char* tess_evaluation_shader_file);
    operator GLuint() const{return program_;}
    void SetLight(const PointLight & light) const;
    void SetModelMatrix(const Mat4 &) const;
    void SetViewMatrix(const Mat4 &) const;
    void SetProjectionMatrix(const Mat4 &) const;
    void SetMaterial(const Material &material) const;
    void SetLambda(GLfloat lambda) const;
    ~Program();
```

```
private:
    GLuint program_;
    GLuint vertex_shader_;
    GLuint fragment_shader_;
    GLuint tess_control_shader_;
    GLuint tess_evaluation_shader_;

  GLuint LoadAndCompileShaderOrDie(
      const char* source_file, GLenum type
      );
  GLuint LinkProgramOrDie(GLint vertex_shader,
      GLint fragment_shader,
      GLint tess_control_shader,
      GLint tess_evaluation_shader
      );
  GLint GetUniformLocationOrDie(const char* var_name);
```

Lokacje zmiennych uniform (prywatne)


```
GLuint lambda_location_;
GLuint model_matrix_location_;
GLuint projection_matrix_location_;
GLuint view_matrix_location_;
struct {
    GLuint emission;
    GLuint ambient;
    GLuint diffuse;
    GLuint specular;
    GLuint shininess;
} material_locations_;
struct {
    GLuint ambient;
    GLuint attenuation;
    GLuint diffuse;
    GLuint position;
    GLuint specular;
}light_locations_;
```

Klasa Bezier. Dane prywatne


```
class Bezier{
  private:
    GLuint vao_;
    GLuint vertex_buffer_;
    Mat4 model_matrix_;
    Material material_;
    float velocity_;
    float angle_;
    bool animating_;
```

- Brak `normal_matrix_` — wektory normalne są obliczane w shaderze

```
class Bezier{
  public:
    Bezier();
    void Initialize(const GLfloat points [64][4]);
    void Draw(const Program & program) const;
    ~Bezier();
    void SetMaterial(const Material & m)
                                    {material_ = m;}
    void Move(float dt);
    void SpeedUp(){velocity_ *= 1.09544511501;}
    void SlowDown(){velocity_/=1.09544511501;}
    void ToggleAnimated(){animating_ = !animating_;}
```

```
void Bezier::Initialize(const GLfloat points[16][4]){
  glGenVertexArrays(1,&vao_);
  glBindVertexArray(vao_);

  glGenBuffers(1, &vertex_buffer_);
  glBindBuffer(GL_ARRAY_BUFFER, vertex_buffer_);
  glBufferData(GL_ARRAY_BUFFER,
       64*sizeof(GLfloat), points, GL_STATIC_DRAW);
  glVertexAttribPointer(0, 4, GL_FLOAT,
        GL_FALSE, 0, (GLvoid*)0);
  glEnableVertexAttribArray(0);

  glBindBuffer(GL_ARRAY_BUFFER, 0);
  glBindVertexArray(0);
}
```

Wyświetlenie. Pierwszy bok


```
void Bezier::Draw(const Program &program) const{
    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);
```

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

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

Dane dla materiału


```
const Material  kYellowMaterial={
    {0.6f, 0.6f, 0.6f, 1.0f}, //Ambient
    {1.0f, 1.0f, 0.0f, 1.0f}, //Diffuse
    {0.6f, 0.6f, 0.6f, 1.0f}, //Specular
    {0.0f, 0.0f, 0.0f, 1.0f}, //Emission
    60.0f //shininess
  };
```

Dane dla światła


```
const PointLight kPointLight={
    {0.0f, 9.5f, 3.0f, 1.0f}, //position
    {0.1f, 0.1f, 0.1f, 1.0f}, //ambient
    {1.0f, 1.0f, 1.0f, 1.0f}, //diffuse
    {1.0f, 1.0f, 1.0f, 1.0f}, //specular
    {0.5f, 0.005f, 0.0125f}     //attenuation
};
```

Dane dla płata


```
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}
};
```