

PGR - Vertex Shader transformace

Tomáš Milet

Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2. 602 00 Brno - Královo Pole
imilet@fit.vutbr.cz



23. října 2019

Geometry Shader

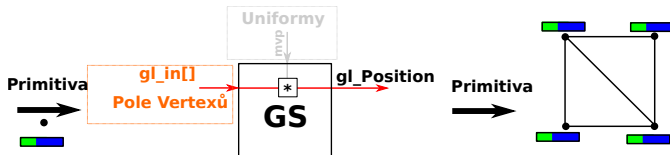
- Geometry shader se nachází za vertex shaderem (za teselací).
- Pracuje po primitivech - má přístup ke všem atributům všech vrcholů vstupního primitiva
- Umožňuje generování geometrie a její úpravu.
- Transformaci bodu na polygon.
- Používá se pro různé efekty (např. stíny (pomocí stínových těles)).
- Další využití může být v částicových systémech.
- Geometry Instancing.
- Transform feedback.

- V Geometry shaderu je nutné specifikovat typ vstupního primitiva.

```
layout (points, invocations=N) in; //vstupni primitivum bude bod  
//points, lines, lines_adjacency, triangles, triangles_adjacency  
//invocations - kolikrat bude GS spusten na jedno primitivum
```

- Také je nutné definovat výstupní primitivum a maximální počet výstupních vertexů.

```
layout (triangle_strip, max_vertices=4) out; //vystup je sekvence troj.  
//points, line_strip, triangle_strip
```



```
#version 430
```

```
layout(points) in;
```

```
layout(triangle_strip,max_vertices=4) out;
```

```
void main() {
```

```
    gl_Position=mvp*(gl_in[0].gl_Position+vec4(-1,-1,0,0));
```

```
    EmitVertex();
```

```
    gl_Position=mvp*(gl_in[0].gl_Position+vec4(-1,+1,0,0));
```

```
    EmitVertex();
```

```
    gl_Position=mvp*(gl_in[0].gl_Position+vec4(+1,-1,0,0));
```

```
    EmitVertex();
```

```
    gl_Position=mvp*(gl_in[0].gl_Position+vec4(+1,+1,0,0));
```

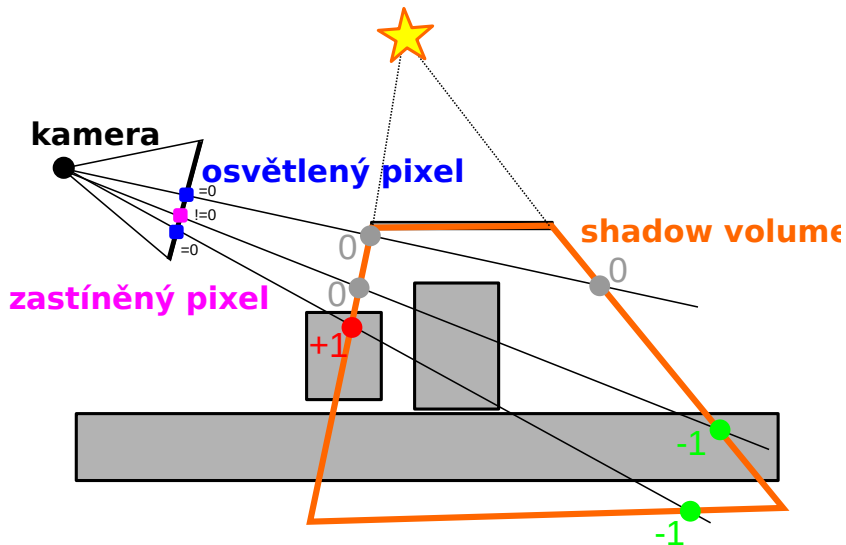
```
    EmitVertex();
```

```
    EndPrimitive();
```

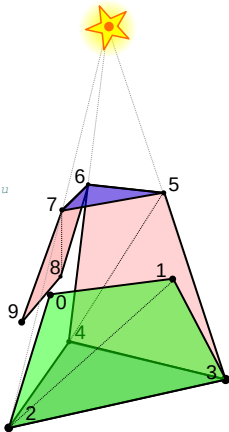
```
}
```

```
#version 430
layout(points) in;
layout(triangle_strip,max_vertices=4) out;
void main() {
    gl_Position=vec4(-1,-1,0,1);EmitVertex();
    gl_Position=vec4(-1,+1,0,1);EmitVertex();
    gl_Position=vec4(+1,-1,0,1);EmitVertex();
    gl_Position=vec4(+1,+1,0,1);EmitVertex();
    EndPrimitive();
}
```

```
glGenVertexArrays(1,&emptyVAO);
//...
glBindVertexArray(emptyVAO);//aktivujeme VAO
glDrawArrays(GL_POINTS,0,1);
glBindVertexArray(0);//deaktivujeme VAO
```

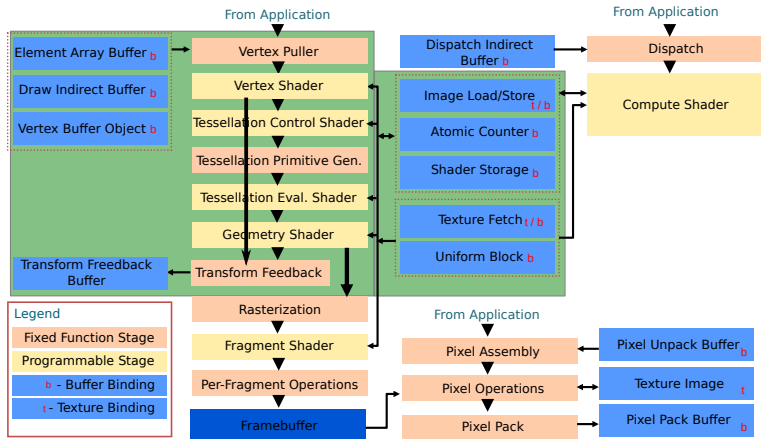


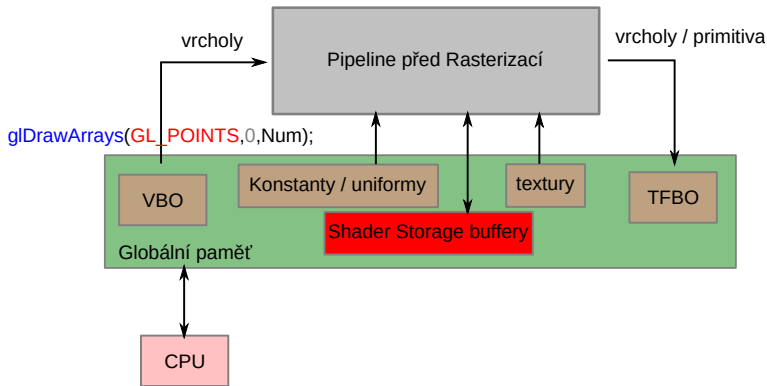
```
#version 330
layout(triangles)in;
layout(triangle_strip,max_vertices=10)out;
uniform mat4 MVP,M; //matice
uniform vec4 LightPosition; //pozice svetla
void main() {
    vec4 LP=M*LightPosition;
    vec4 p[6];
    p[0]=gl_in[0].gl_Position; //body trojuhelniku
    p[1]=gl_in[1].gl_Position;
    p[2]=gl_in[2].gl_Position;
    p[3]=vec4(gl_in[0].gl_Position.xyz*LP.w-LP.xyz,0); //v nekonecnu
    p[4]=vec4(gl_in[1].gl_Position.xyz*LP.w-LP.xyz,0);
    p[5]=vec4(gl_in[2].gl_Position.xyz*LP.w-LP.xyz,0);
    vec3 N=normalize(cross((p[1]-p[0]).xyz,(p[2]-p[0]).xyz));
    float Distance=dot(N,LP.xyz)-dot(N,p[0].xyz);
    if(Distance<=0){ //otocime volume vnitkem ven
        vec4 c=p[0];p[0]=p[1];p[1]=c;
        c=p[3];p[3]=p[4];p[4]=c;
    }
    gl_Position=MVP*p[0];EmitVertex();
    gl_Position=MVP*p[1];EmitVertex();
    gl_Position=MVP*p[3];EmitVertex();
    gl_Position=MVP*p[4];EmitVertex();
    gl_Position=MVP*p[5];EmitVertex();
    gl_Position=MVP*p[1];EmitVertex();
    gl_Position=MVP*p[2];EmitVertex();
    gl_Position=MVP*p[0];EmitVertex();
    gl_Position=MVP*p[5];EmitVertex();
    gl_Position=MVP*p[3];EmitVertex();
    EndPrimitive();
}
```



Transform feedback

- Zápis primitiv do bufferu
- Hlavně v Geometry Shaderu
- Streams (kreslení i zápis o bufferu)





```
const char*Vayrings[]={ "Out1", "Out2"};
glTransformFeedbackVaryings (Program,2,Varyings, GL_SEPARATE_ATTRIBS);
glLinkProgram (Program);

//...

glBindBufferBase (GL_TRANSFORM_FEEDBACK_BUFFER,0,Buffer1);
glBindBufferBase (GL_TRANSFORM_FEEDBACK_BUFFER,1,Buffer2);

glEnable (GL_RASTERIZER_DISCARD);//nebudeme rasterizovat
//...
glBeginTransformFeedback (GL_TRIANGLES);
glDrawArrays (...);
glEndTransformFeedback ();
```

Slinkovat program s nastavenými výstupními proměnnými v shaderu. C++:

```
//seznam promennych v shaderu, ktere se budou pomoci TF zapisovat do bufferu
const char*ResetVaryings[]={ "vPosition", "vVelocity", "vMass" };
//nastavime seznam a nastavime prokladany zapis
glTransformFeedbackVaryings(ResetProgram,3,ResetVaryings, GL_INTERLEAVED_ATTRIBS)
//znovu slinkujeme program
glLinkProgram(ResetProgram);
```

glsl:

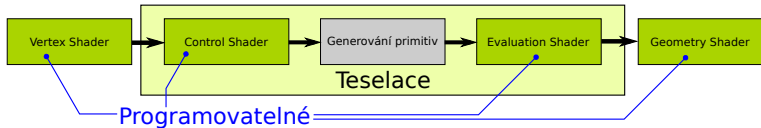
```
#version 330

layout(location=0) out vec2  vPosition; //pozice castice
layout(location=1) out vec2  vVelocity; //rychlost castice
layout(location=2) out float vMass; //hmotnost castice
//...
void main() {
    vPosition = vec2(0); //pozice do prostred
    vVelocity = vec2(cos(VelAngle), sin(VelAngle)) * VelSize; //rychlost jako vektor
    vMass      = Noise(MassSeed + uint(gl_VertexID), MinMass, MaxMass); //hmotnost
}
```

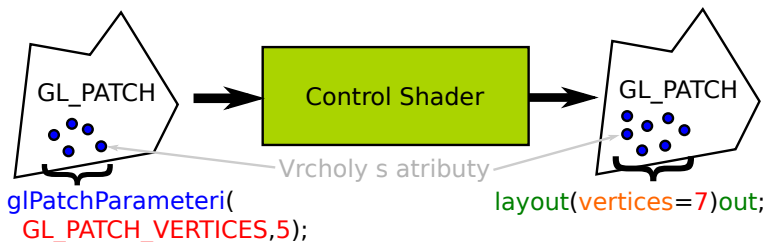
Teselace

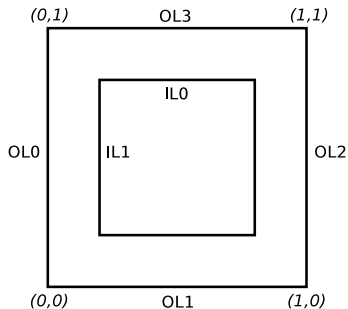
- Teselace je rozřezání jednoho primitiva na více spojených.
- Může se použít pro zjemnění geometrie
- Nachází se za vertex shaderem a před geometry shaderem.
- Složená ze 3 částí:
 - Control Shader
 - Generování primitiv/Teselace
 - Evaluation Shader
- Nový typ primitiva **GL_PATCHES**

```
glPatchParameteri(GL_PATCH_VERTICES,10);//nastavi pocet vrcholu patche - 10  
glDrawArrays(GL_PATCHES,0,100);//vykresli 10 patchu po 10 vrcholech
```

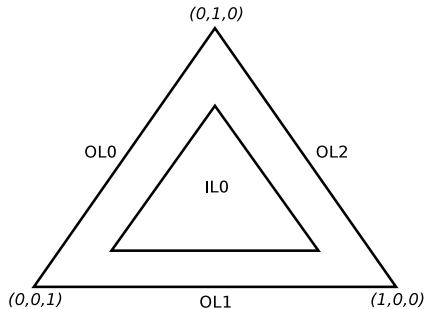


- Řídí stupěň teselace
- Počítá kontrolní body
- Je spouštěn tolikrát, kolik je vertexů ve výstupním primitivu
- Číslo spuštění uloženo v `gl_InvocationID`
- `barrier()`





Quads



Triangles

- `layout ({isolines,triangles,quads}) in;`
- `gl_TessLevelOuter(4),gl_TessLevelInner(2)`

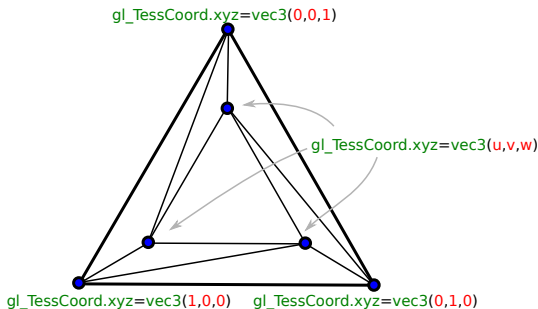
```
#version 430

// pocet vertexu ve vystupni primitivu
// pocet spusteni control shaderu
layout(vertices=3) out;

uniform vec2 TessLevelInner;//vnitrni deleni
uniform vec4 TessLevelOuter;//deleni hran

void main() {
    //velikost gl_in zavisi na GL_PATCH_VERTICES
    //velikost gl_out zavisi na layout(vertices=n)out;
    gl_out[gl_InvocationID].gl_Position=gl_in[gl_InvocationID].gl_Position;
    if(gl_InvocationID==0){
        gl_TessLevelOuter[0]=TessLevelOuter[0];
        gl_TessLevelOuter[1]=TessLevelOuter[1];
        gl_TessLevelOuter[2]=TessLevelOuter[2];
        gl_TessLevelOuter[3]=TessLevelOuter[3];
        gl_TessLevelInner[0]=TessLevelInner[0];
        gl_TessLevelInner[1]=TessLevelInner[1];
    }
}
```

- Nastavuje typ primitiva **isolines, triangles, quads**
- Počítá souřadnice vrcholů nateselovaného primitiva
- Souřadnice do primitiva **gl_TessCoord**
- je spoštěn pro každý nateselovaný vrchol
- vygenerovaná primitiva jdou dále do geometry shaderu



- Nateselovaný čtyřúhelník
- výpočet pozic vrcholů

```
#version 430
```

```
layout (quads) in;
```

```
void main() {
```

```
    vec4 A=mix(gl_in[0].gl_Position,gl_in[1].gl_Position,gl_TessCoord.x);
```

```
    vec4 B=mix(gl_in[3].gl_Position,gl_in[2].gl_Position,gl_TessCoord.x);
```

```
    gl_Position=mix(A,B,gl_TessCoord.y);
```

```
}
```

```
// Vertex shader
#version 430
void main() {
    gl_Position = mvp*position;
}

// Control shader
#version 430
layout(vertices=16) out;

void main() {
    gl_out[gl_InvocationID].gl_Position =
    gl_in[gl_InvocationID].gl_Position;
    if(gl_InvocationID == 0) {
        gl_TessLevelInner[0] = gl_TessLevelInner[1] =
        gl_TessLevelOuter[0] = gl_TessLevelOuter[1] =
        gl_TessLevelOuter[2] = gl_TessLevelOuter[3] = 64;
    }
}
```

```
// Evaluation shader
#version 430
layout(quads, ccw) in;

vec4 bernstein(float t) {
    return vec4((1-t)*(1-t)*(1-t), 3*t*(1-t)*(1-t), 3*t*t*(1-t), t*t*t);
}

void main() {
    vec4 bu = bernstein(gl_TessCoord.x);
    vec4 bv = bernstein(gl_TessCoord.y);
    vec4 position = vec4(0, 0, 0, 0);
    for(int y = 0; y < 4; ++y){
        for(int x = 0; x < 4; ++x){
            position += bu[x]*bv[y]*gl_in[4*y + x].gl_Position;
        }
    }
    gl_Position = position;
}
```

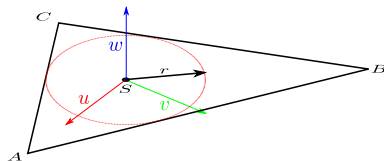
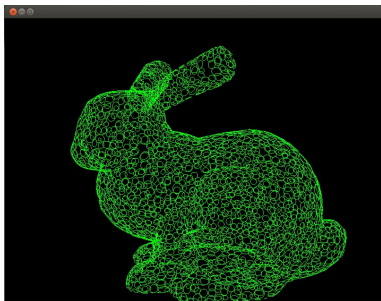
```
#version 430

//vertex shader
out vec4 vAttrib;
gl_Position

//control shader
//atribut z vertex shaderu jejich pocet je rizen glPatchParameteri(GL_PATCH_VERTICES,n);
in vec4 vAttrib[]; //atribut z vertex shaderu
gl_in[].gl_Position; //atribut pozice z vertex shaderu
//atribut z control shaderu do evaluation shaderu, pocet je rizen pomoci layout(vertices=n)out;
out vec4 cAttrib[]; //atribut pro vrchol z control shaderu do evaluation shaderu
//per patch atribut z control shaderu do evaluation shaderu, pocet je 1
patch out mat4 cM; //atribut pro patch z control shaderu do evaluation shaderu

//evaluation shader
in vec4 cAttrib[];
patch in mat4 cM;
//atribut z evaluation shaderu do geometry shaderu
out vec3 eNormal;

//geometry shader
//pocet je rizen typem primitiva
in vec3 eNormal[];
```



$$K = \begin{pmatrix} 1 & 0 & 0 & S_x \\ 0 & 1 & 0 & S_y \\ 0 & 0 & 1 & S_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} r & 0 & 0 & 0 \\ 0 & r & 0 & 0 \\ 0 & 0 & r & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} u_x & v_x & w_x & 0 \\ u_y & v_y & w_y & 0 \\ u_z & v_z & w_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Control Shader

```
#version 400
layout(vertices=1)out;
patch out mat4 K;
void main() {
    gl_TessLevelOuter[0]=1;
    gl_TessLevelOuter[1]=64;
    gl_TessLevelOuter[2]=1;
    gl_TessLevelOuter[3]=1;
    gl_TessLevelInner[0]=1;
    gl_TessLevelInner[1]=1;
    vec4 TT[3];
    TT[0]=gl_in[0].gl_Position;
    TT[1]=gl_in[1].gl_Position;
    TT[2]=gl_in[2].gl_Position;
    float t01=length((TT[0]-TT[1]).xyz);
    float t02=length((TT[0]-TT[2]).xyz);
    float t12=length((TT[1]-TT[2]).xyz);
    float s=t01+t02+t12;
    float r=sqrt((s/2-t01)*(s/2-t02)*(s/2-t12)*s/2)*2/s;
    t01/=s;
    t02/=s;
    t12/=s;
    vec3 C=TT[0].xyz*t12+TT[1].xyz*t02+TT[2].xyz*t01;
    vec3 x=normalize(TT[0].xyz-C);
    vec3 y=normalize(TT[1].xyz-C);
    vec3 z=normalize(cross(x,y));
    y=normalize(cross(z,x));
    K=mat4(vec4(x,0)*r,vec4(y,0)*r,vec4(z,0)*r,vec4(C,1));
}
```

Evaluation Shader

```
#version 400

#define MY_PI 3.14159265359

layout(isolines)in;

uniform mat4 V;
uniform mat4 P;

patch in mat4 K;

void main(){
    float Angle=gl_TessCoord.x*MY_PI*2;
    vec4 PP=vec4(cos(Angle),sin(Angle),0,1);
    gl_Position=P*V*K*PP;
}
```

Outer level:

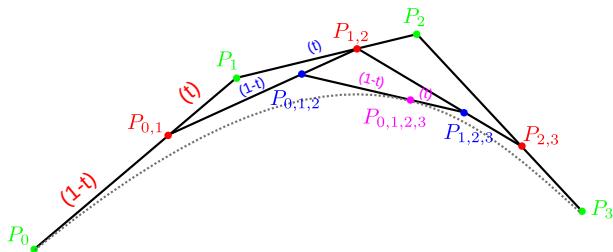
- Strany ploch musí odpovídat (zamezení T-spojů)
- Transformovat kontrolní body na obrazovku
- Spočítat delků hran
- Dělit maximální delkou hrany

Inner level:

- Z přílušných outer-levelů
- průměr, maximum, ...
- Korekce podle vnitřních kontrolních bodů

Ukázka v aplikaci

$$\vec{P}_{0,1,2,3}(t) = (1, t^1, t^2, t^3) \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{pmatrix} \cdot \begin{pmatrix} \vec{P}_0 \\ \vec{P}_1 \\ \vec{P}_2 \\ \vec{P}_3 \end{pmatrix}$$



$$\begin{aligned}
 \vec{P}_{0,1} &= (1-t)\vec{P}_0 + (t)\vec{P}_1 \\
 \vec{P}_{1,2} &= (1-t)\vec{P}_1 + (t)\vec{P}_2 \\
 \vec{P}_{2,3} &= (1-t)\vec{P}_2 + (t)\vec{P}_3 \\
 \vec{P}_{0,1,2} &= (1-t)\vec{P}_{0,1} + (t)\vec{P}_{1,2} \\
 \vec{P}_{1,2,3} &= (1-t)\vec{P}_{1,2} + (t)\vec{P}_{2,3} \\
 \vec{P}_{0,1,2,3} &= (1-t)\vec{P}_{0,1,2} + (t)\vec{P}_{1,2,3}
 \end{aligned}$$

$$\begin{aligned}
 \vec{P}_{0,1} &= (1-t)\vec{P}_0 + (t)\vec{P}_1 \\
 \vec{P}_{1,2} &= (1-t)\vec{P}_1 + (t)\vec{P}_2 \\
 \vec{P}_{2,3} &= (1-t)\vec{P}_2 + (t)\vec{P}_3 \\
 \vec{P}_{0,1,2} &= (1-t)\vec{P}_{0,1} + (t)\vec{P}_{1,2} \\
 \vec{P}_{1,2,3} &= (1-t)\vec{P}_{1,2} + (t)\vec{P}_{2,3} \\
 \vec{P}_{0,1,2,3} &= (1-t)\vec{P}_{0,1,2} + (t)\vec{P}_{1,2,3}
 \end{aligned}$$

$$\begin{aligned}
 \vec{P}_{0,1,2} &= (1-t)((1-t)\vec{P}_0 + (t)\vec{P}_1) + (t)((1-t)\vec{P}_1 + (t)\vec{P}_2) \\
 \vec{P}_{1,2,3} &= (1-t)((1-t)\vec{P}_1 + (t)\vec{P}_2) + (t)((1-t)\vec{P}_2 + (t)\vec{P}_3) \\
 \vec{P}_{0,1,2,3} &= (1-t)\vec{P}_{0,1,2} + (t)\vec{P}_{1,2,3}
 \end{aligned}$$

$$\begin{aligned}
 \vec{P}_{0,1,2,3} &= (1-t) \begin{pmatrix} (1-t) & (1-t) & (1-t) \\ (1-t) & (1-t) & (t) \\ (1-t) & (t) & (1-t) \\ (1-t) & (t) & (t) \end{pmatrix} \begin{pmatrix} \vec{P}_0 \\ \vec{P}_1 \\ \vec{P}_1 \\ \vec{P}_2 \end{pmatrix} + \\
 &\quad \begin{pmatrix} (t) & (1-t) & (1-t) \\ (t) & (1-t) & (t) \\ (t) & (t) & (1-t) \\ (t) & (t) & (t) \end{pmatrix} \begin{pmatrix} \vec{P}_1 \\ \vec{P}_2 \\ \vec{P}_2 \\ \vec{P}_3 \end{pmatrix} +
 \end{aligned}$$

$$\begin{aligned}
 \vec{P}_{0,1,2,3} &= (1-t) \begin{pmatrix} (1-t) & (1-t) & (1-t) \\ 3(1-t) & (1-t) & (t) \\ 3(1-t) & (t) & (t) \\ (t) & (t) & (t) \end{pmatrix} \begin{pmatrix} \vec{P}_0 \\ \vec{P}_1 \\ \vec{P}_2 \\ \vec{P}_3 \end{pmatrix} +
 \end{aligned}$$

$$\vec{P}_{0,1,2,3} = \begin{pmatrix} (1-t) \\ 3(1-t) \\ 3(1-t) \\ (t) \end{pmatrix} \begin{pmatrix} (1-t) \\ (1-t) \\ (t) \\ (t) \end{pmatrix} \begin{pmatrix} (1-t) \\ (t) \\ (t) \\ (t) \end{pmatrix} \begin{pmatrix} \vec{P}_0 \\ \vec{P}_1 \\ \vec{P}_2 \\ \vec{P}_3 \end{pmatrix} + + + +$$

$$\vec{P}_{0,1,2,3} = \begin{pmatrix} (1-t)^3 \\ 3(1-t)^2(t) \\ 3(1-t)(t)^2 \\ (t)^3 \end{pmatrix} \begin{pmatrix} \vec{P}_0 \\ \vec{P}_1 \\ \vec{P}_2 \\ \vec{P}_3 \end{pmatrix} + + + +$$

$$\vec{P}_{0,1,2,3} = \begin{pmatrix} 3 \\ 0 \\ 1 \\ 3 \\ 2 \\ 3 \end{pmatrix} \begin{pmatrix} (1-t)^{3-0}(t)^0 \\ (1-t)^{3-1}(t)^1 \\ (1-t)^{3-2}(t)^2 \\ (1-t)^{3-3}(t)^3 \end{pmatrix} \begin{pmatrix} \vec{P}_0 \\ \vec{P}_1 \\ \vec{P}_2 \\ \vec{P}_3 \end{pmatrix} + + + +$$

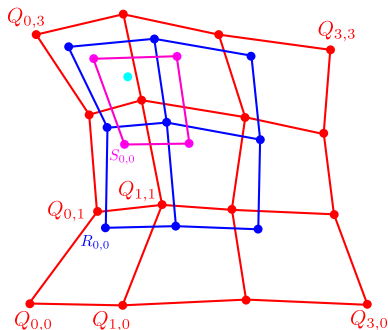
$$\vec{P}_{0,1,2,3}(t) = \sum_{i=0}^{i \leq 3} \binom{3}{i} (1-t)^{3-i} (t)^i \vec{P}_i$$

$$\vec{P}(t) = \sum_{i=0}^{i \leq n-1} \binom{n-1}{i} (1-t)^{n-1-i} (t)^i \vec{P}_i$$

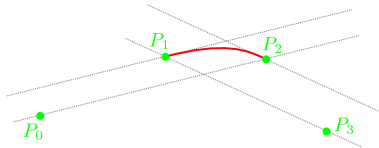
$$\vec{P}_{0,1,2,3}(t) = \begin{matrix} (1-t)^3 \\ 3(1-t)^2(t) \\ 3(1-t)(t)^2 \\ (t)^3 \end{matrix} \begin{matrix} \vec{P}_0 \\ \vec{P}_1 \\ \vec{P}_2 \\ \vec{P}_3 \end{matrix} +$$

$$\vec{P}_{0,1,2,3}(t) = \begin{matrix} (1-3t+3t^2-t^3) \\ (3t-6t^2+3t^3) \\ (3t^2-3t^3) \\ (t^3) \end{matrix} \begin{matrix} \vec{P}_0 \\ \vec{P}_1 \\ \vec{P}_2 \\ \vec{P}_3 \end{matrix} +$$

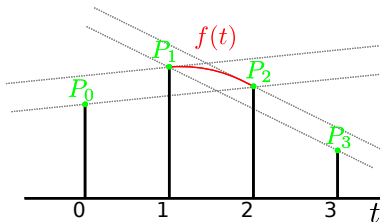
$$\vec{P}_{0,1,2,3}(t) = (1, t^1, t^2, t^3) \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{pmatrix} \cdot \begin{pmatrix} \vec{P}_0 \\ \vec{P}_1 \\ \vec{P}_2 \\ \vec{P}_3 \end{pmatrix}$$



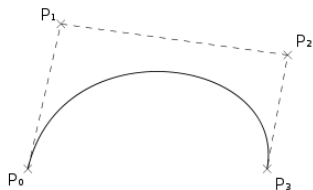
$$\begin{aligned}
 R_{0,0} &= (1 - \eta)(1 - t)Q_{0,0} + (1 - \eta)tQ_{1,0} + (\eta)(1 - t)Q_{0,1} + (\eta)tQ_{1,1} \\
 R_{1,0} &= (1 - \eta)(1 - t)Q_{1,0} + (1 - \eta)tQ_{2,0} + (\eta)(1 - t)Q_{1,1} + (\eta)tQ_{2,1} \\
 R_{2,0} &= (1 - \eta)(1 - t)Q_{2,0} + (1 - \eta)tQ_{3,0} + (\eta)(1 - t)Q_{2,1} + (\eta)tQ_{3,1} \\
 &\vdots \\
 R_{i,k} &= (1 - \eta)(1 - t)Q_{i,k} + (1 - \eta)tQ_{i+1,k} + (\eta)(1 - t)Q_{i,k+1} + (\eta)tQ_{i+1,k+1} \\
 R_{j,k} &= (1 - \eta)(1 - t)Q_{j,k} + (1 - \eta)tQ_{j+1,k} + (\eta)(1 - t)Q_{j,k+1} + (\eta)tQ_{j+1,k+1} \\
 S_{j,k} &= (1 - \eta)(1 - t)R_{j,k} + (1 - \eta)tR_{j+1,k} + (\eta)(1 - t)R_{j,k+1} + (\eta)tR_{j+1,k+1} \\
 X_{j,k} &= (1 - \eta)(1 - t)S_{j,k} + (1 - \eta)tS_{j+1,k} + (\eta)(1 - t)S_{j,k+1} + (\eta)tS_{j+1,k+1}
 \end{aligned}$$



$$v(t) = (1, t^1, t^2, t^3) \cdot \frac{1}{2} \cdot \begin{pmatrix} 0 & 2 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 2 & -5 & 4 & -1 \\ -1 & 3 & -3 & 1 \end{pmatrix} \cdot \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix}$$



$$\begin{aligned}
 f(0) &= P_1 \\
 f(1) &= P_2 \\
 f'(0) &= \frac{P_2 - P_0}{2} \\
 f'(1) &= \frac{P_3 - P_1}{2} \\
 f(t) &= \sum_{i=0}^{i \leq 3} (a_i + b_i t + c_i t^2 + d_i t^3) P_i
 \end{aligned}$$



$$B_{n,k}(t) = \binom{n}{k} t^k (1-t)^{(n-k)}$$

$$P(t) = \left[(B_{n,0}(t), \dots, B_{n,n}(t)) \begin{pmatrix} x_0 & y_0 & \dots \\ x_1 & y_1 & \dots \\ \dots & \dots & \dots \end{pmatrix} \right]$$

$$P_{\{x,y,z\}}(u,v) = \left[(B_{n,0}(u), \dots, B_{n,n}(u)) K_{\{x,y,z\}} \begin{pmatrix} B_{n,0}(v) \\ \dots \\ B_{n,n}(v) \end{pmatrix} \right]$$

Kontrolní body mají i váhu :

$$P(t) = \frac{\sum_{i=0}^n B_{i,n}(t) \mathbf{P}_i \mathbf{w}_i}{\sum_{i=0}^n B_{i,n}(t) \mathbf{w}_i}$$

Kontrolní body mají i váhu :

$$P(t) = \frac{\sum_{i=0}^n B_{i,n}(t) \mathbf{P}_i \mathbf{w}_i}{\sum_{i=0}^n B_{i,n}(t) \mathbf{w}_i}$$

- Váha táhne plochu blíž k bodu.
- Kontrolní body s vahou jsou homogenní souřadnice.
- Jdou bez problémů i promítnout.

```
#version 410

uniform mat4 mvp;

layout(location=0) in vec4 position;

void main()
{
    gl_Position = mvp*position;
}
```

```
#version 410

flat in vec4 color;

out vec4 output;

void main()
{
    output = color;
}
```

```
#version 410

layout(vertices=16) out;

void main()
{
    gl_out[gl_InvocationID].gl_Positiono
        = gl_in[gl_InvocationID].gl_Position;
    if(gl_InvocationID == 0)
    {
        gl_TessLevelInner[0] = 8;
        gl_TessLevelInner[1] = 8;
        gl_TessLevelOuter[0] = 64;
        gl_TessLevelOuter[1] = 64;
        gl_TessLevelOuter[2] = 64;
        gl_TessLevelOuter[3] = 64;
    }
}
```

```
#version 410
layout(quads, equal_spacing, ccw) in;
flat out vec4 color;
vec4 bernstein(float t)
{
    return vec4((1-t)*(1-t)*(1-t), 3*t*(1-t)*(1-t), 3*t*t*(1-t), t*t*t);
}
void main()
{
    vec4 bu = bernstein(gl_TessCoord.x);
    vec4 bv = bernstein(gl_TessCoord.y);
    vec4 position = vec4(0,0,0,0);
    for(int y = 0; y < 4; ++y)
        for(int x = 0; x < 4; ++x)
            position += bu[x]*bv[y]*gl_in[4*y+x].gl_Position;
    gl_Position = position;
    color = vec4(gl_TessCoord.xy, 0, 1);
}
```

- <http://www.opengl.org/sdk/docs/>
- <http://www.opengl.org/documentation/glsl/>
- <http://www.opengl.org/registry/>

Thank you for your attention! Questions?