Samuel LE BERRE

TP 10 : Shaders

Scaling

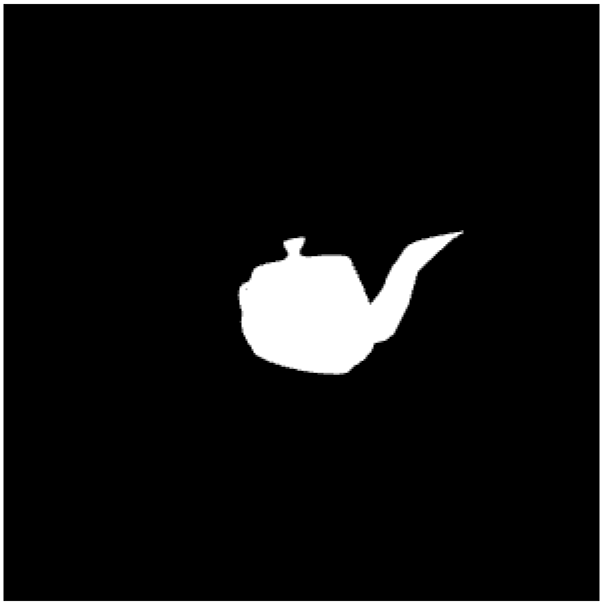
Vertex Shader

```
void main(void) {
   vec4 v = vec4(vertex,5.0);
   v.x = v.x * 2.0;  //agrandit de 2 foiss la taille en x
   v.z = v.z * 0.5; //diminue par 2 la taille en z
   gl_Position = _mvProj * v; //execute le scaling
}
```

Fragment Shader

```
void main(void)
{
   gl_FragColor = vec4(0.0, 1.0, 0.0, 1.0); // met l'objet en vert
}
```

Image



Wave

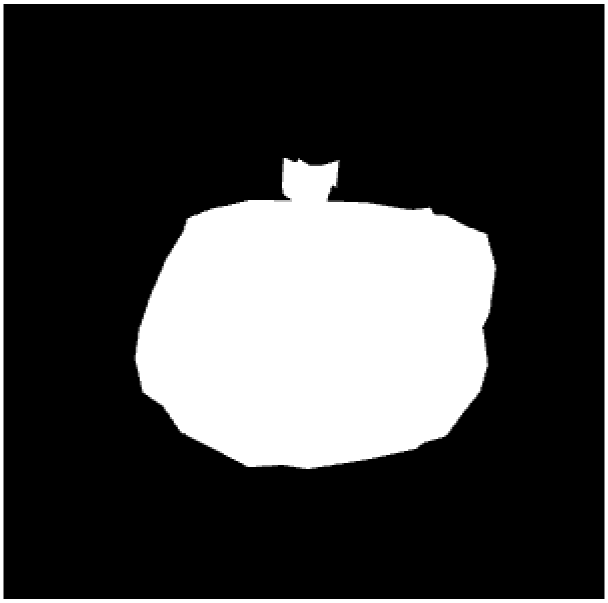
Vertex Shader

```
void main(void) {
   vec4 v = vec4(vertex,1.0);
   float s = 1.0 + 0.1*sin(v.s*_time)*sin(v.z*_time);
   v.y = s * v.y;
   gl_Position = _mvProj * v;
}
```

Fragment Shader

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

Image



Particule

Vertex Shader

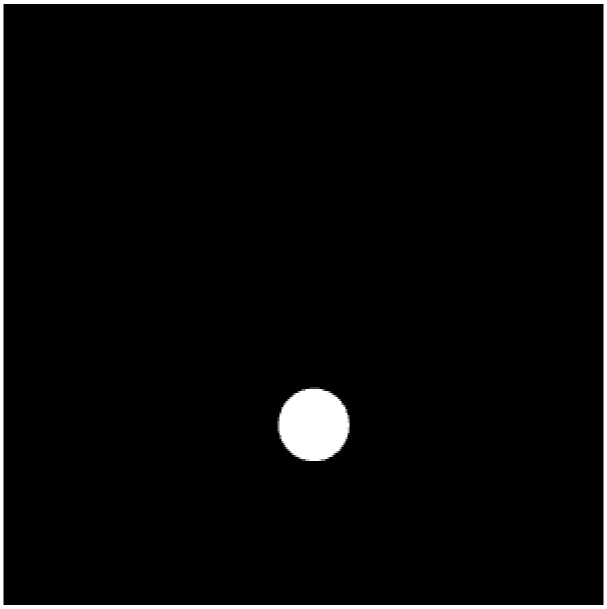
```
void main(void) {
   float g = 0.0981;
   float m = 1.0;
   vec3 object_pos;
   vec3 velocite = vec3(0.2,-0.7,0.1);
```

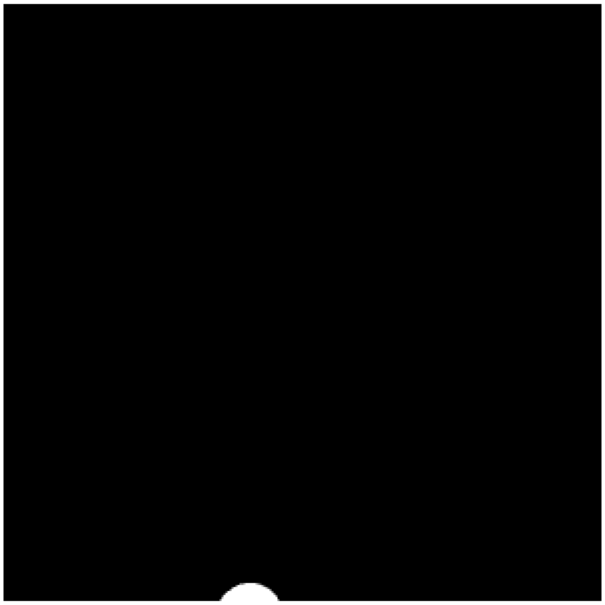
```
    object_pos.x = vertex.x + velocite.x*_time/5000.0;
    object_pos.y = vertex.y + velocite.y*_time/5000.0 - g/m*_time/5000.0*_time/5000.0;
    object_pos.z = vertex.z + velocite.z*_time/5000.0;
    gl_Position = _mvProj * vec4(object_pos,7.0);

}
```

Fragment Shader

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

Image





Gouraud

Vertex Shader

```
uniform mat4 _mvProj;
uniform mat3 _norm;
uniform mat4 _mv;
uniform float _time; // time in seconds
#pragma include "light.glsl"
varying vec4 color;

void main(void) {
   float f; /* compute normalized normal, light vector, view vector,      half-angle vector in eye
coordinates */
   vec3 norm = normalize( _norm * normal);
   vec3 mvcoordonne = vec3(_mv[0]);
         vec3 lightv = normalize(vec3(0.0,0.0,2.0) - (mvcoordonne * vertex).xyz);
         vec3 viewv = -normalize((mvcoordonne * vertex).xyz);
         vec3 halfv = normalize(lightv * viewv);

   if (dot(lightv, norm) > 0.0)
         f = 1.0;
   else
         f = 0.0;

   vec3 diffuse3;
         float specular;
         float glowingSpecular = 50.0;
         getDirectionalLight(norm, _dLight, glowingSpecular, diffuse3, specular);
         float shininess = 1.0;

   vec4 ambient4 = vec4(vec3(1.0,1.0,1.0) * _dLight[0],1.0);
   vec4 diffuse4 = vec4(diffuse3 * _dLight[1] * dot(lightv,norm),1.0);
   vec4 specular4 = vec4(vec3(specular * _dLight[2] * pow(dot(lightv,halfv), shininess)),1.0);
   color = f * (ambient4 + diffuse4 + specular4);
   color.a = 1.0;
   gl_Position = _mvProj * vec4(vertex,2.0);
}
```

Fragment Shader

```
varying vec4 color;
void main(void)
{
   gl_FragColor = vec4(color);
}
```

Image

Pas fonctionnelle

Phong

Vertex Shader

```
varying vec3 normale;
varying vec3 positione;

void main(void) {
        normale = _norm * normal;
        positione = vec3(_mv[2]) * vertex;
        gl_Position = _mvProj * vec4(vertex,1.0);
}
```

Fragment Shader

```
varying vec3 normale;
varying vec3 positione;
#pragma include "light.glsl"

void main(void)
{
        vec3 norm = normalize(normale);
        vec3 lightv = normalize(vec3(0.0,0.0,2.0) - positione);
        vec3 viewv = normalize(positione);
        vec3 halfv = normalize(lightv + viewv);

         vec3 diffuse3;
         float specular;
         float glowingSpecular = 50.0;
         getDirectionalLight(norm, _dLight, glowingSpecular, diffuse3, specular);
         float shininess = 1.0;

        vec4 diffuse4 = vec4(max(0.0, dot(lightv, viewv))*diffuse3*_dLight[1],1.0);
        vec4 ambient4 = vec4(vec3(1.0,1.0,1.0) * _dLight[0],1.0);
        vec3 specular3 = vec3(pow(max(0.0, dot(norm, halfv)), shininess) * specular * _dLight[2]);
   vec3 color = vec3(ambient4.xyz + diffuse4.xyz + specular3);
   gl_FragColor = vec4(color, 1.0);
}
```

Image