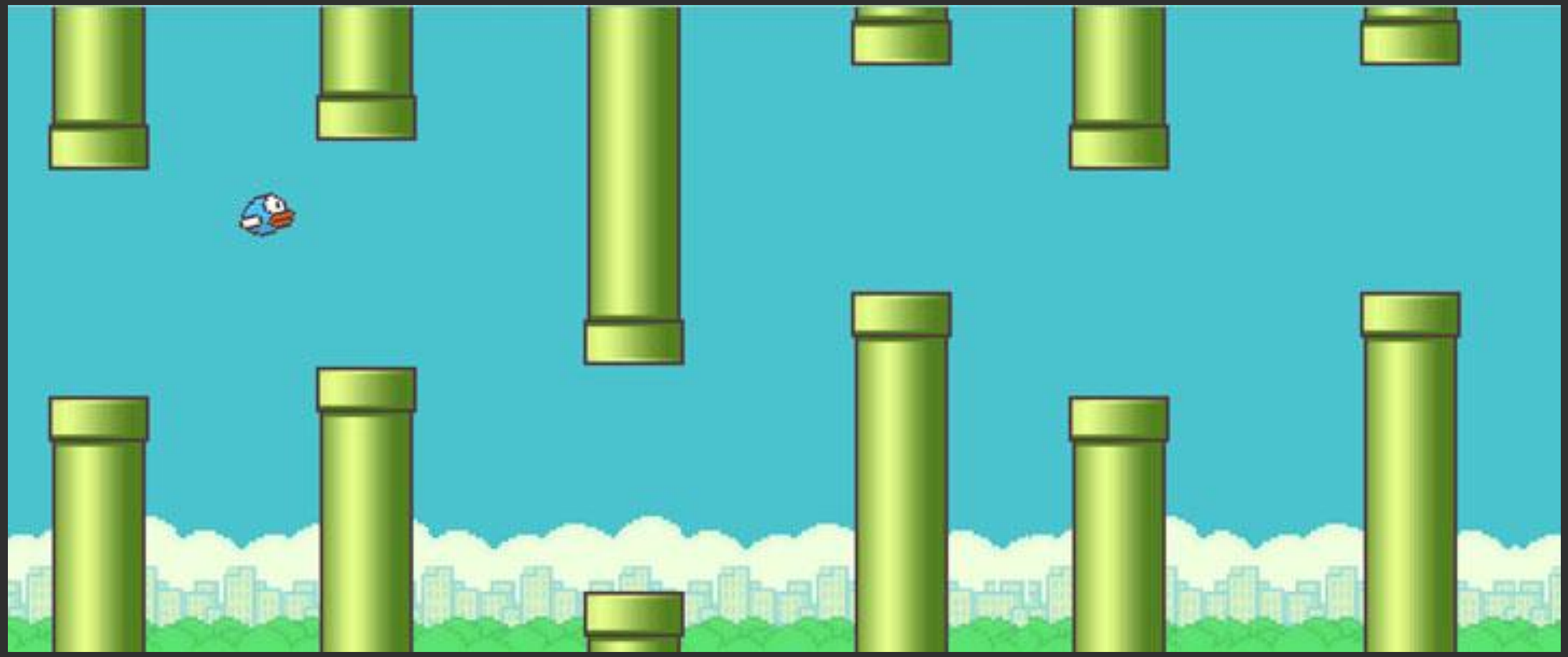
Ballwall „3D – Flappy Bird“

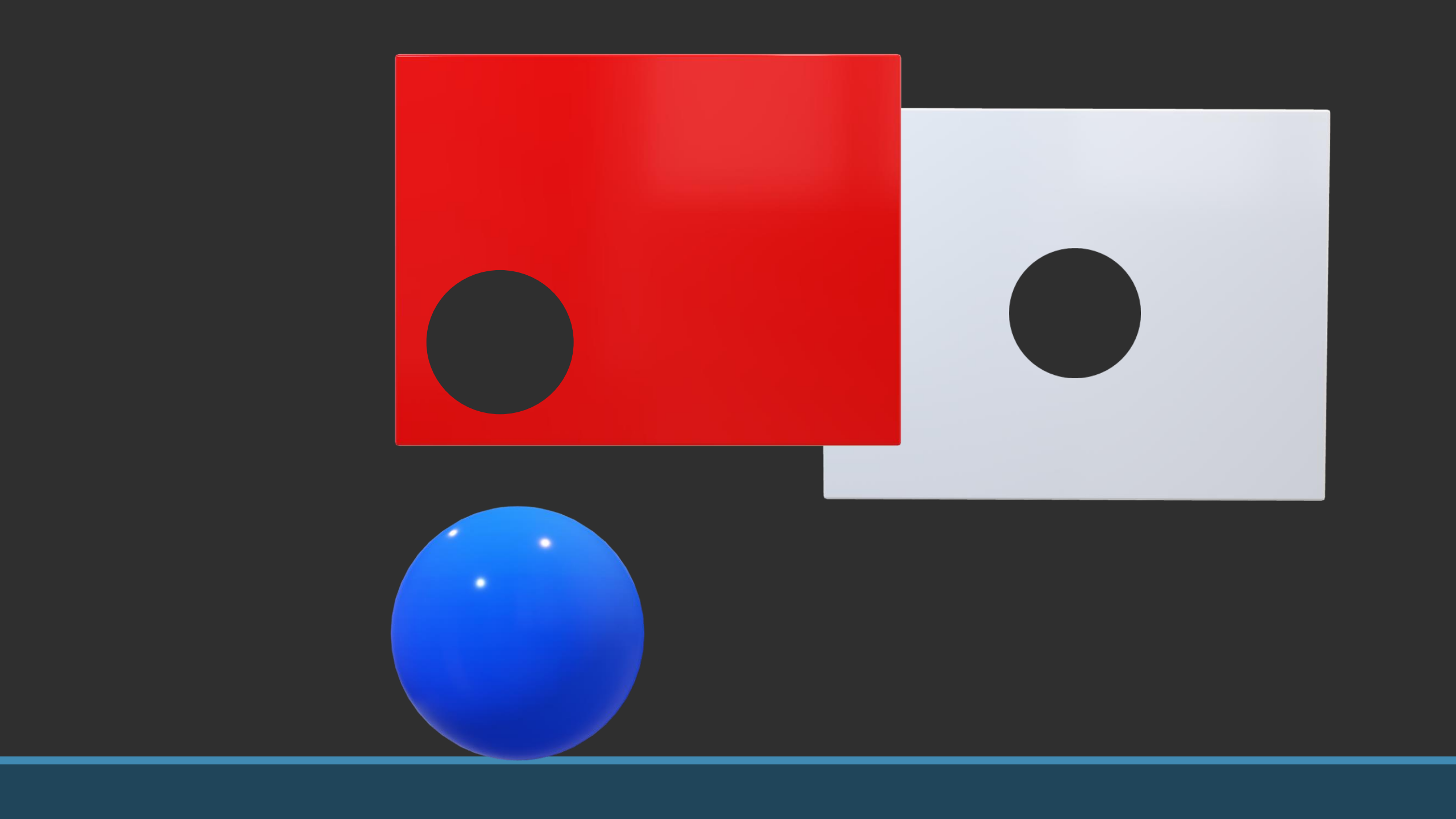
Samardzic Nuris, 01611855

Tosun Begüm, 11719058

Contents

- Basic Idea
- Libraries
- Code





Libraries

- glew v2.2.0 OpenGL Extension Wrangler Library (GLEW) :
  - a cross-platform open-source C/C++ extension loading library
  - provides efficient run-time mechanisms for determining which OpenGL extensions are supported on the target platform
  - currently maintained by Nigel Stewart with bug fixes, new OpenGL extension support and new releases
  - was developed by Milan Ikits and Marcelo Magallon
  - Aaron Lefohn, Joe Kniss, and Chris Wyman were the first users and also assisted with the design and debugging process

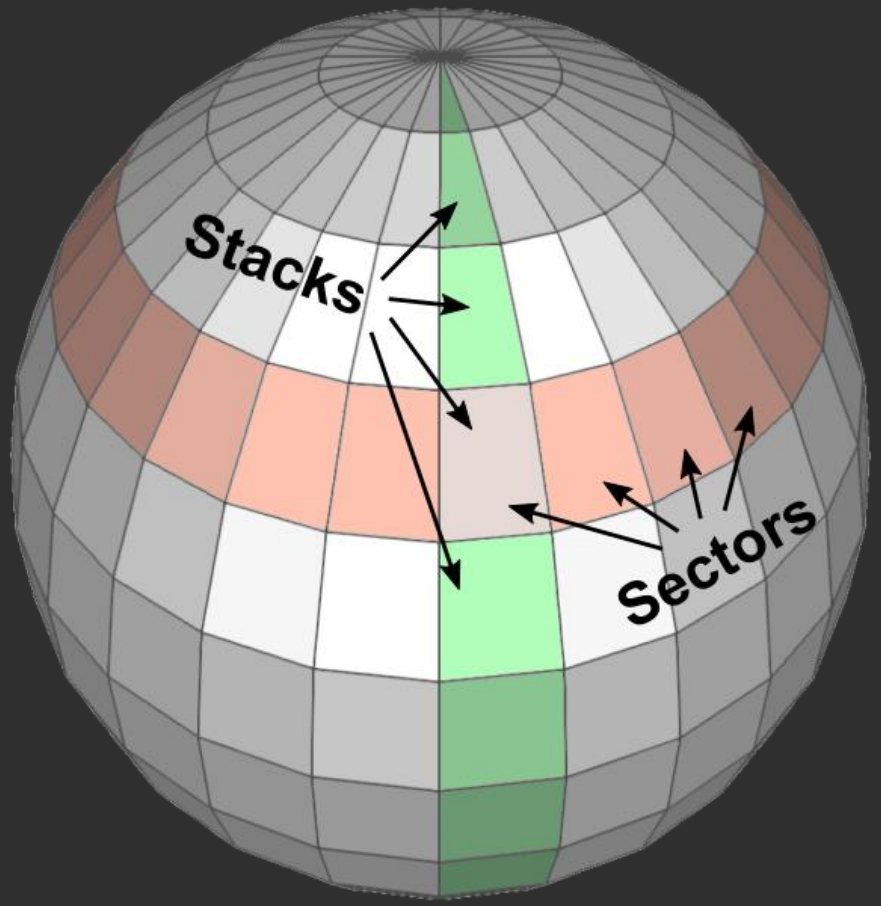
Libraries

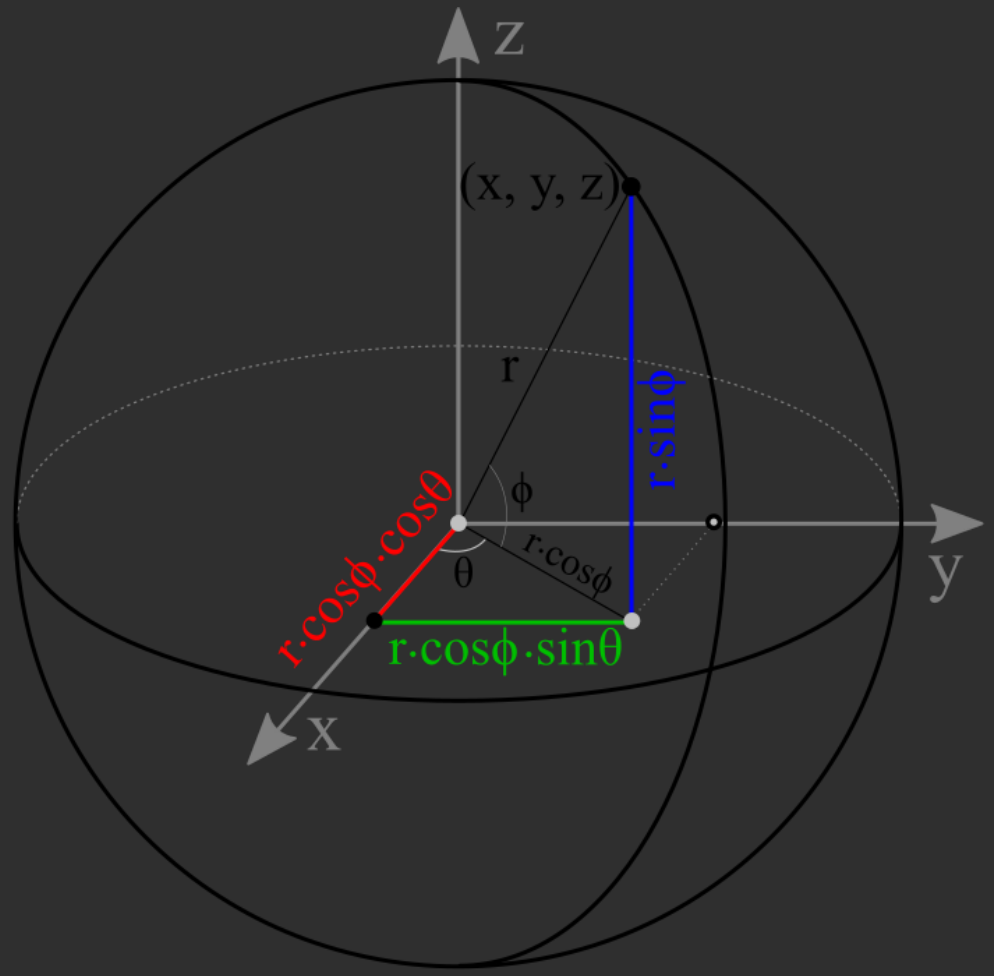
- freeglut v3.0.0 free GL Utility Toolkit:
  - GLUT vs freeglut:
  - GLUT is getting old and really needs improvement
  - its license does not allow anyone to distribute modified library code

  - freeglut is a free-software/open-source alternative to the GLUT library
  - originally written by Pawel W. Olszta with contributions from Andreas Umbach and Steve Baker
  - John F. Fay, John Tsiombikas, and Diederick C. Niehorster are the current maintainers of the freeglut project

Sphere

- 3D closed surface where every point on the sphere is same distance (radius) from a given point
- $x^2 + y^2 + z^2 = r^2$
- we cannot draw all the points on a sphere
- sample a limited amount of points by dividing the sphere by sectors and stacks
- an arbitrary point (x, y, z) on a sphere can be computed with the corresponding sector angle θ and stack angle ϕ
- $$x = (r \cdot \cos\phi) \cdot \cos\theta \qquad z = r \cdot \sin\phi$$
  $$y = (r \cdot \cos\phi) \cdot \sin\theta$$


Stacks
Sectors


z
(x, y, z)
r
r·sinϕ
ϕ
r·cosϕ·cosθ
θ
r·cosϕ
y
r·cosϕ·sinθ
x

Sphere

- range of sector angles is from 0 to 360 degrees
- stack angles are from 90 (top) to -90 degrees (bottom)
-

$$\theta = 2\pi \cdot \frac{\text{sectorStep}}{\text{sectorCount}}$$

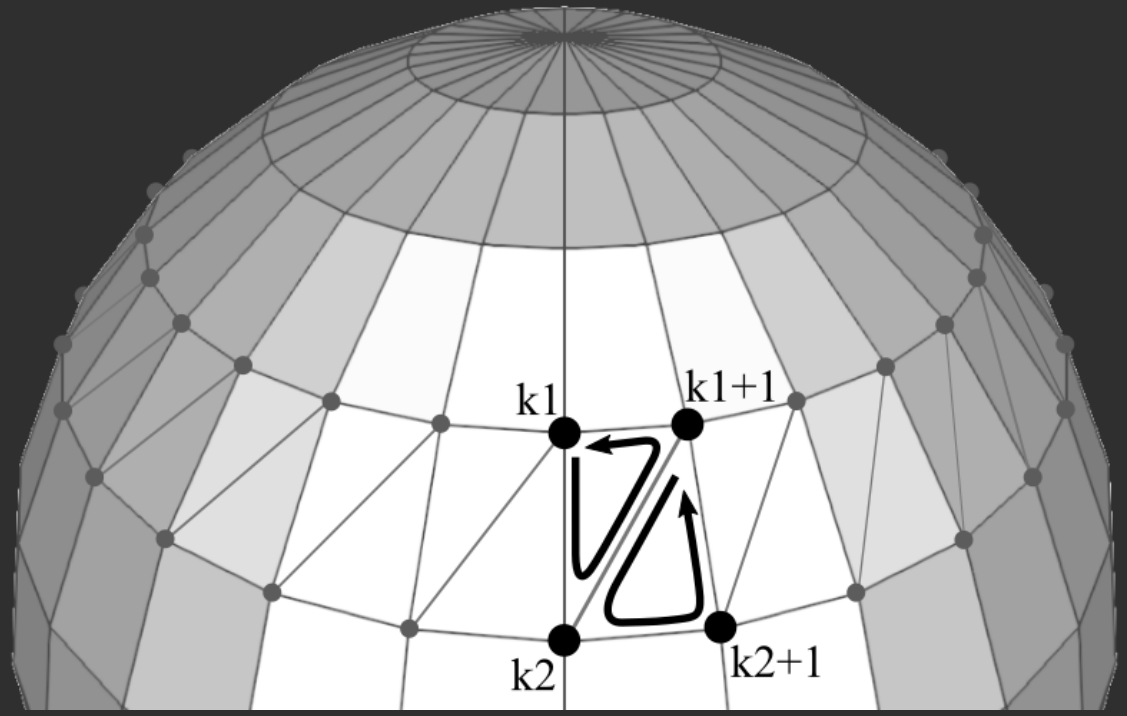
$$\phi = \frac{\pi}{2} - \pi \cdot \frac{\text{stackStep}}{\text{stackCount}}$$

Creating the sphere

- adapted from http://www.songho.ca/opengl/gl_sphere.html
- done in three main steps

Creating the sphere: 1.step

```
271         for(int i = 0; i <= stackCount; ++i)
272         {
273             stackAngle = Pi / 2 - i * stackStep;        // starting from pi/2 to -pi/2
274             xy = radius * cosf(stackAngle);             // r * cos(u)
275             z = radius * sinf(stackAngle);              // r * sin(u)
276 
277             // add (sectorCount+1) vertices per stack
278             // the first and last vertices have same position and normal, but different tex coords
279             for(int j = 0; j <= sectorCount; ++j)
280             {
281                 sectorAngle = j * sectorStep;           // starting from 0 to 2pi
282 
283                 // vertex position (x, y, z)
284                 x = xy * cosf(sectorAngle);             // r * cos(u) * cos(v)
285                 y = xy * sinf(sectorAngle);             // r * cos(u) * sin(v)
286                 vnbuffer2.push_back(x);
287                 vnbuffer2.push_back(y);
288                 vnbuffer2.push_back(z);
289 
290                 // normalized vertex normal (nx, ny, nz)
291                 nx = x * lengthInv;
292                 ny = y * lengthInv;
293                 nz = z * lengthInv;
294                 vnbuffer2.push_back(nx);
295                 vnbuffer2.push_back(ny);
296                 vnbuffer2.push_back(nz);
297             }
298         }
```



Sphere

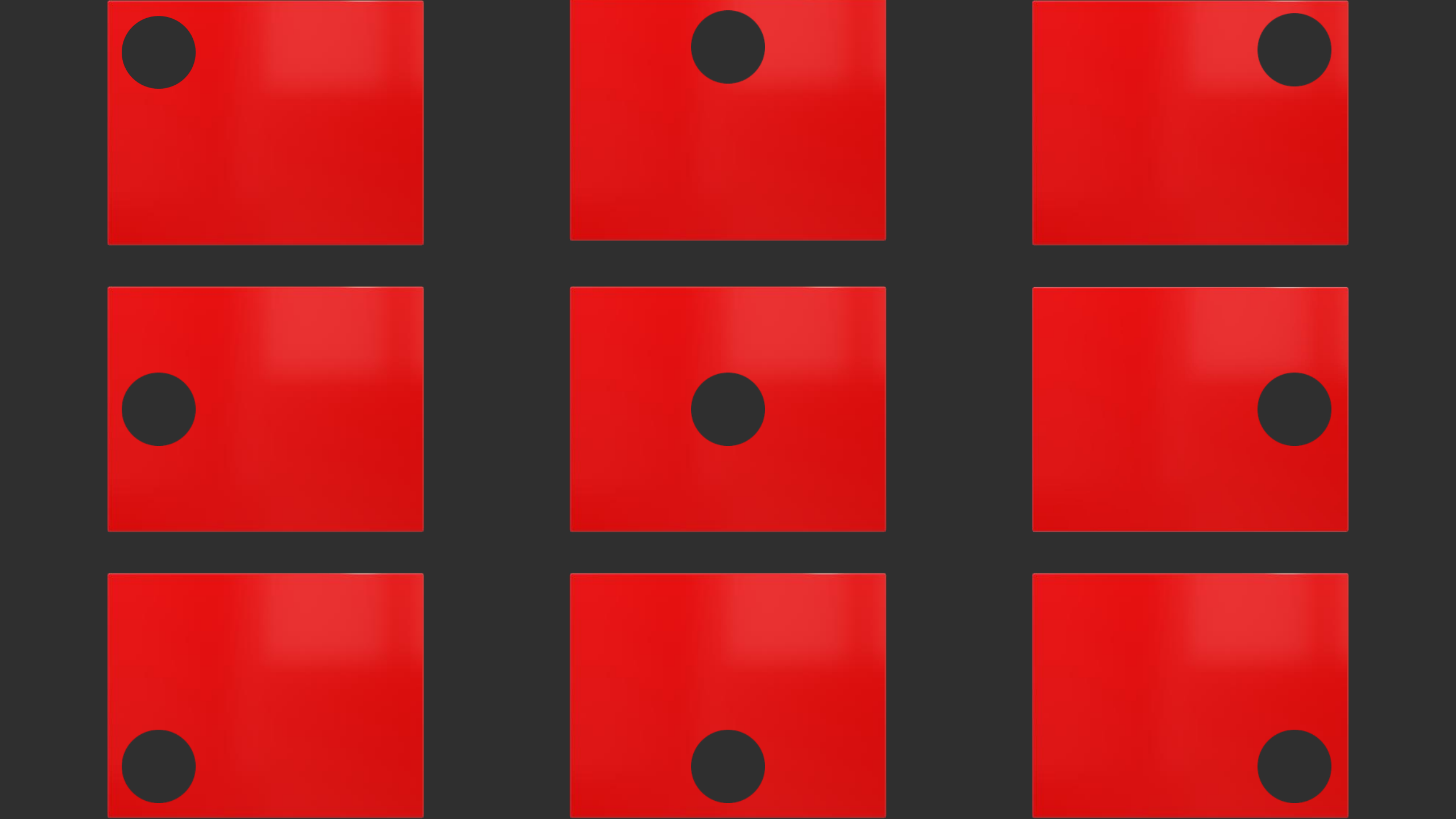
- to draw the surface of a sphere in OpenGL, you must triangulate adjacent vertices to form polygons
- Each sector in a stack requires 2 triangles
- If the first vertex index in the current stack is k1 and the next stack is k2, then the counterclockwise orders of vertex indices of 2 triangles are;
- k1 ⟶ k2 ⟶ k1+1
- k1+1 ⟶ k2 ⟶ k2+1

Creating the sphere: 2.step

```
306         for(int i = 0; i < stackCount; ++i)
307         {
308             k1 = i * (sectorCount + 1);     // beginning of current stack
309             k2 = k1 + sectorCount + 1;      // beginning of next stack
310
311             for(int j = 0; j < sectorCount; ++j, ++k1, ++k2)
312             {
313                 // 2 triangles per sector excluding first and last stacks
314                 // k1 => k2 => k1+1
315                 if(i != 0)
316                 {
317                         indices.push_back(k1);
318                         indices.push_back(k2);
319                         indices.push_back(k1 + 1);
320                 }
321
322                 // k1+1 => k2 => k2+1
323                 if(i != (stackCount-1))
324                 {
325                         indices.push_back(k1 + 1);
326                         indices.push_back(k2);
327                         indices.push_back(k2 + 1);
328                 }
329             }
330         }
```

```
331         GLuint VBO, EBO;
332         glGenVertexArrays(1, &sphereId); // generate the vertex array index for the sphere
333         glGenBuffers(1, &VBO); // generate a vertex buffer
334         glGenBuffers(1, &EBO); // generate a index buffer
335         glBindVertexArray(sphereId); // bind the sphere id
336         glBindBuffer(GL_ARRAY_BUFFER, VBO); // bind the vertex buffer
337         // fill the vertex buffer
338         glBufferData(GL_ARRAY_BUFFER, vnbuffer2.size()*sizeof(float), &vnbuffer2[0], GL_STATIC_DRAW);
339         glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO); // bind the index buffer
340         NumSphereIndices = (int)indices.size();
341         // fill the index buffer
342         glBufferData(GL_ELEMENT_ARRAY_BUFFER, NumSphereIndices*sizeof(float), &indices[0], GL_STATIC_DRAW);
343         // specify the location and data format of the vertex buffer array
344         glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6*sizeof(GLfloat), (void*)0);
345         glEnableVertexAttribArray(0);
346         glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6*sizeof(GLfloat), (void*)(3*sizeof(GLfloat)));
347         glEnableVertexAttribArray(1);
348         glBindBuffer(GL_ARRAY_BUFFER, 0);
349         glBindVertexArray(0); // unbind the sphere id
```

Creating the sphere: 3.step



```
54      #define zwallinit 25.0f
55      // set wall parameters
56      void nextwall() {
57          walltype = rand()%9; // wall type
58          zwall = -zwallinit; // initial z position
59          holex = (walltype%3 - 1)*holedist; // x coordinate of the hole center
60          holey = (walltype/3 - 1)*holedist; // y coordinate of the hole center
61          // saturated color:
62          int i = rand()%3, j = rand()%2;
63          wallcolor[i] = 1.0f;
64          wallcolor[(1 + i + j)%3] = 0.0f;
65      //  wallcolor[(1 + i + !j)%3] = (float)rand()/RAND_MAX;
66          wallcolor[(1 + i + !j)%3] = (rand()%5)*0.25f;
67      }
```

Creating walls

```
23    // set the projection matrix (Mp)
24    void camera::perspective(double fovy, double ratiohw, double cnear, double cfar) {
25        double _t = 1/tan(fovy*(M_PI/360)); // tan of half fovy
26        Mp[0][0] =  ratiohw*_t;
27        Mp[1][1] =  _t;
28        Mp[2][2] =  (cfar + cnear)/(cnear - cfar);
29        Mp[2][3] =  2*cfar*cnear/(cnear - cfar);
30        Mp[3][2] = -1;
31        Mp[0][1] = Mp[0][2] = Mp[0][3] = Mp[1][0] = 0;
32        Mp[1][2] = Mp[1][3] = Mp[2][0] = Mp[2][1] = 0;
33        Mp[3][0] = Mp[3][1] = Mp[3][3] = 0;
34    }
35
36
37    void camera::LookAt(double x, double y, double z, double cx, double cy, double cz, double ux, double uy, double uz) {
38        double u[3] = {ux, uy, uz};
39        pos[0] = x; pos[1] = y; pos[2] = z;
40        fwd[0] = cx - x; fwd[1] = cy - y; fwd[2] = cz - z;
41        normalize3d(fwd);
42        crossproduct(right, fwd, u);
43        normalize3d(right);
44        crossproduct(up, right, fwd);
45        calcMatrixModelView();
46    }
```

Perspective / View

Perspective / View

```
64    void camera::calcMatrixModelViewProjection() {
65        for (int j = 0; j != 4; j++)
66            for (int i = 0; i != 4; i++) {
67                Mvp[j][i] = 0;
68                for (int k = 0; k != 4; k++)
69                    Mvp[j][i] += Mp[j][k]*Mv[k][i];
70            }
71    }
```

Shaders

```
#version 330 compatibility
// adapted from the shader code in main.cpp file,
  http://www.songho.ca/opengl/files/sphereShader.zip
layout (location = 0) in vec3 vertexPosition;
layout (location = 1) in vec3 vertexNormal;

uniform mat4 matrixModelViewProjection;
uniform mat4 matrixModelView;
uniform mat3 matrixNormal;
uniform vec3 displacement;

out vec3 esVertex, esNormal;
out float z;

void main()
{
    vec4 newposition = vec4(vertexPosition + displacement, 1.0);
    esVertex = vec3(matrixModelView * newposition);
    esNormal = matrixNormal * vertexNormal;
    z = -newposition.z;
    gl_Position = matrixModelViewProjection * newposition;
}

```

Shaders

```
#version 330 compatibility
// adapted from the shader code in main.cpp file, http://www.songho.ca/opengl/files/sphereShader.zip

uniform vec4 lightPosition;             // should be in the eye space
uniform vec4 lightAmbient;              // light ambient color
uniform vec4 lightDiffuse;              // light diffuse color
uniform vec4 lightSpecular;             // light specular color
uniform vec4 materialColor;             // material ambient and diffuse color
uniform vec4 materialSpecular;          // material specular color
uniform float materialShininess;        // material specular shininess

in vec3 esVertex, esNormal;
in float z;

out vec4 vFragColor;

void main()
{
    vec3 normal = normalize(esNormal);
    vec3 light;
    if(lightPosition.w == 0.0)
    {
        light = normalize(lightPosition.xyz);
    }
    else
    {
        light = normalize(lightPosition.xyz - esVertex);
    }
    vec3 view = normalize(-esVertex);
    vec3 halfv = normalize(light + view);

    vec3 color = lightAmbient.rgb * materialColor.rgb;        // begin with ambient
    float dotNL = max(dot(normal, light), 0.0);

    color += lightDiffuse.rgb * materialColor.rgb * dotNL;    // add diffuse
    float dotNH = max(dot(normal, halfv), 0.0);
    color += pow(dotNH, materialShininess) * lightSpecular.rgb * materialSpecular.rgb; // add specular

        vec3 fogColor = vec3(0.0, 0.0, 0.0);
        color = mix(color, fogColor, smoothstep(-8.0, 27.0, z));

    // set frag color
    vFragColor = vec4(color, materialColor.a);
}

```

Demo