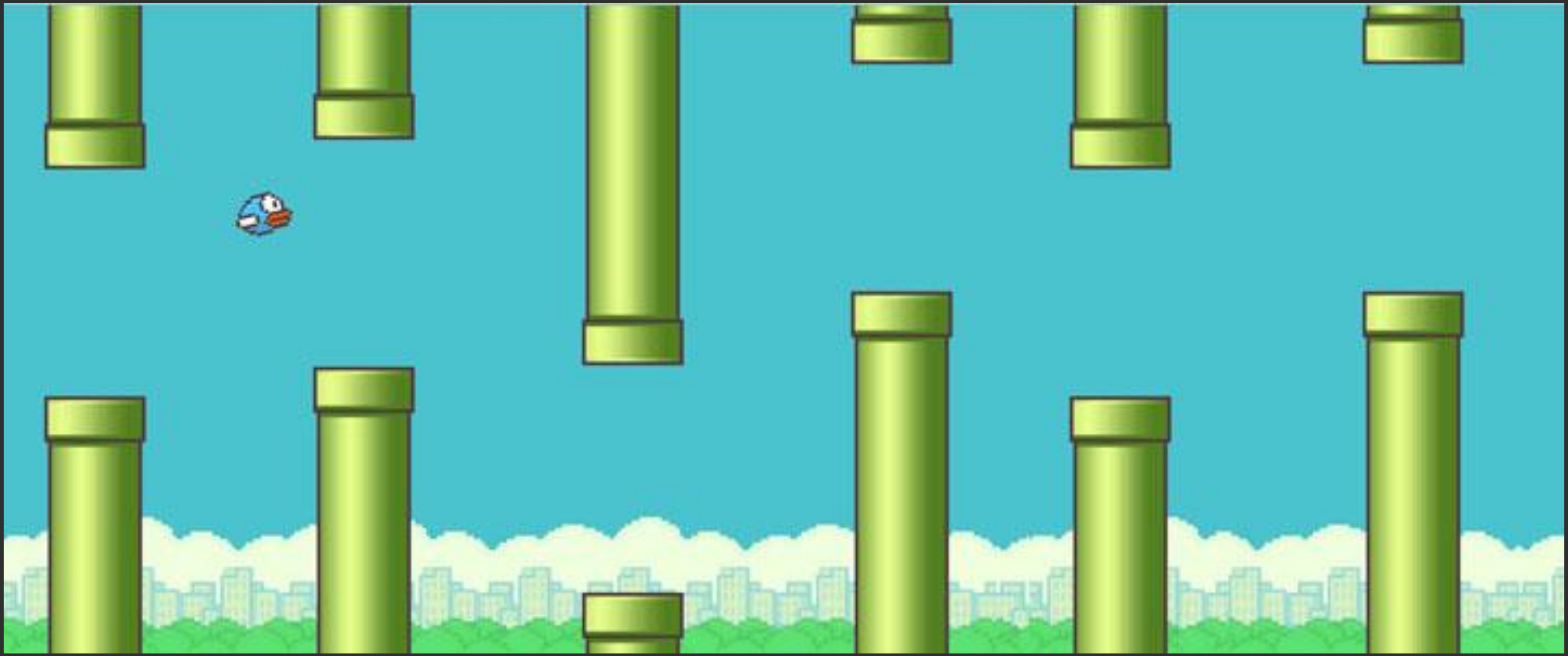
Ballwall
„3D – Flappy Bird“

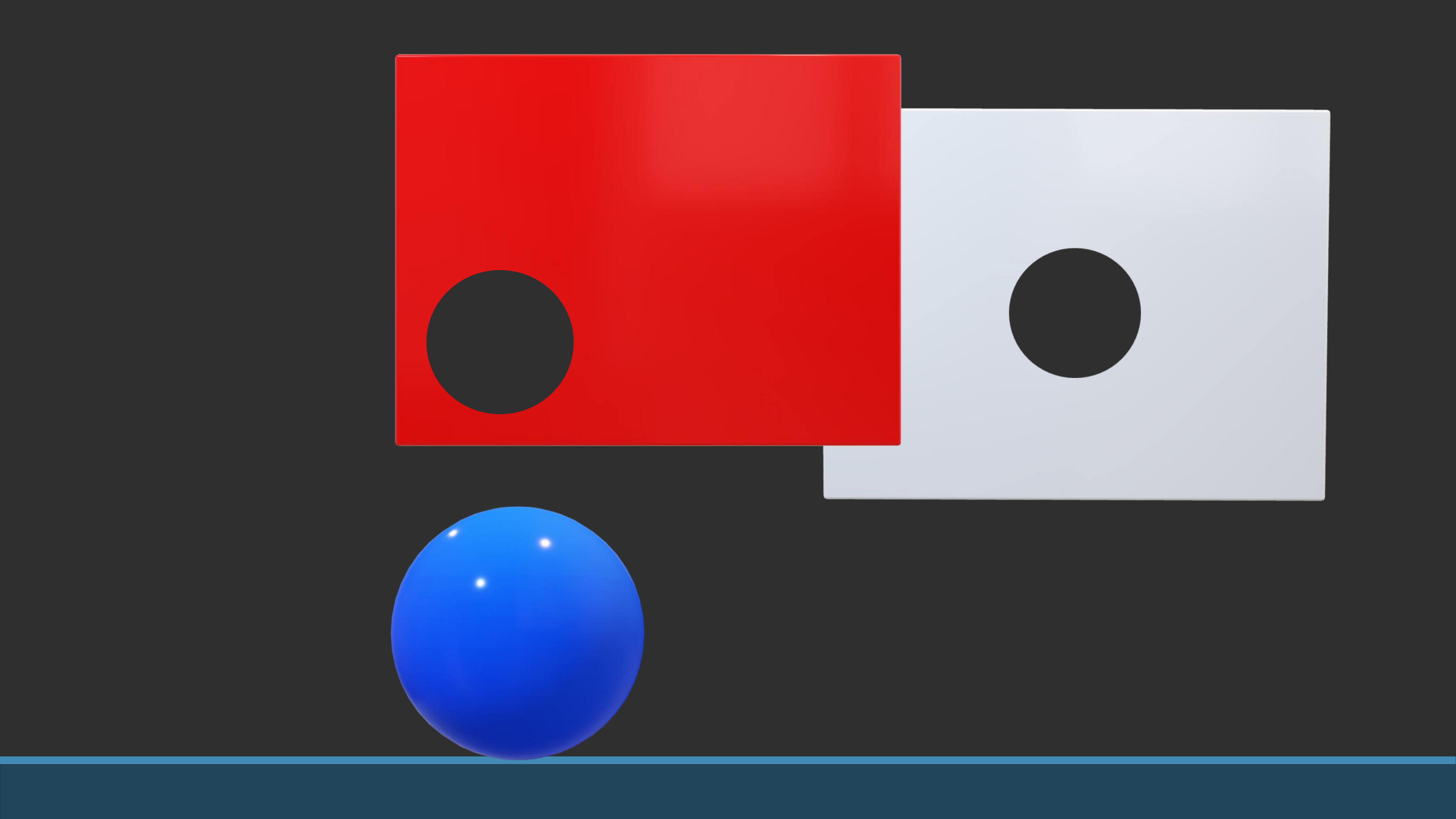
Samardzic Nuris, 01611855

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

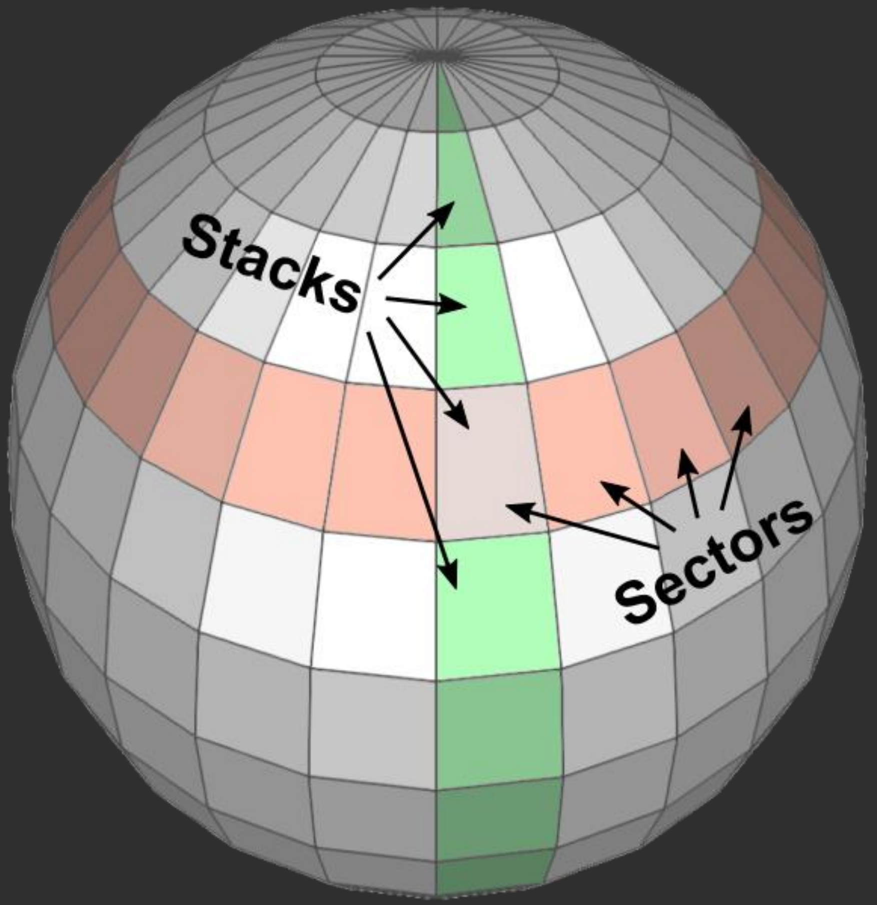
- freeglut v3.0.0 free GL Utility Toolkit:
  - GLUT vs freeglut:
  - GLUT is getting old and really needs improvement
  - its license does not allow anyone to distribute modified library code

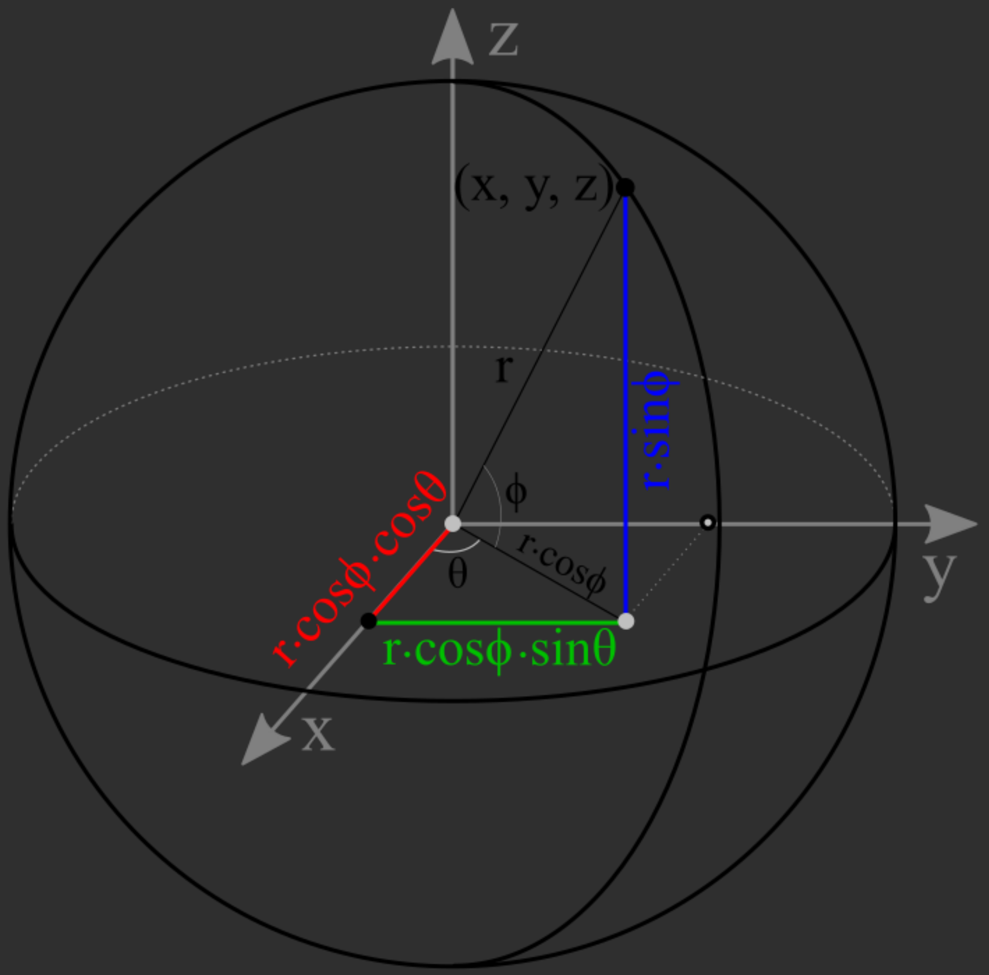
  - freeglut is a free-software/open-source alternative to the GLUT library
  - originally written by Pawel W. Olszta with contributions from Andreas Umbach and Steve Baker
  - John F. Fay, John Tsiombikas, and Diederick C. Niehorster are the current maintainers of the freeglut project

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
r·cosϕ·sinθ
y
x

Sphere

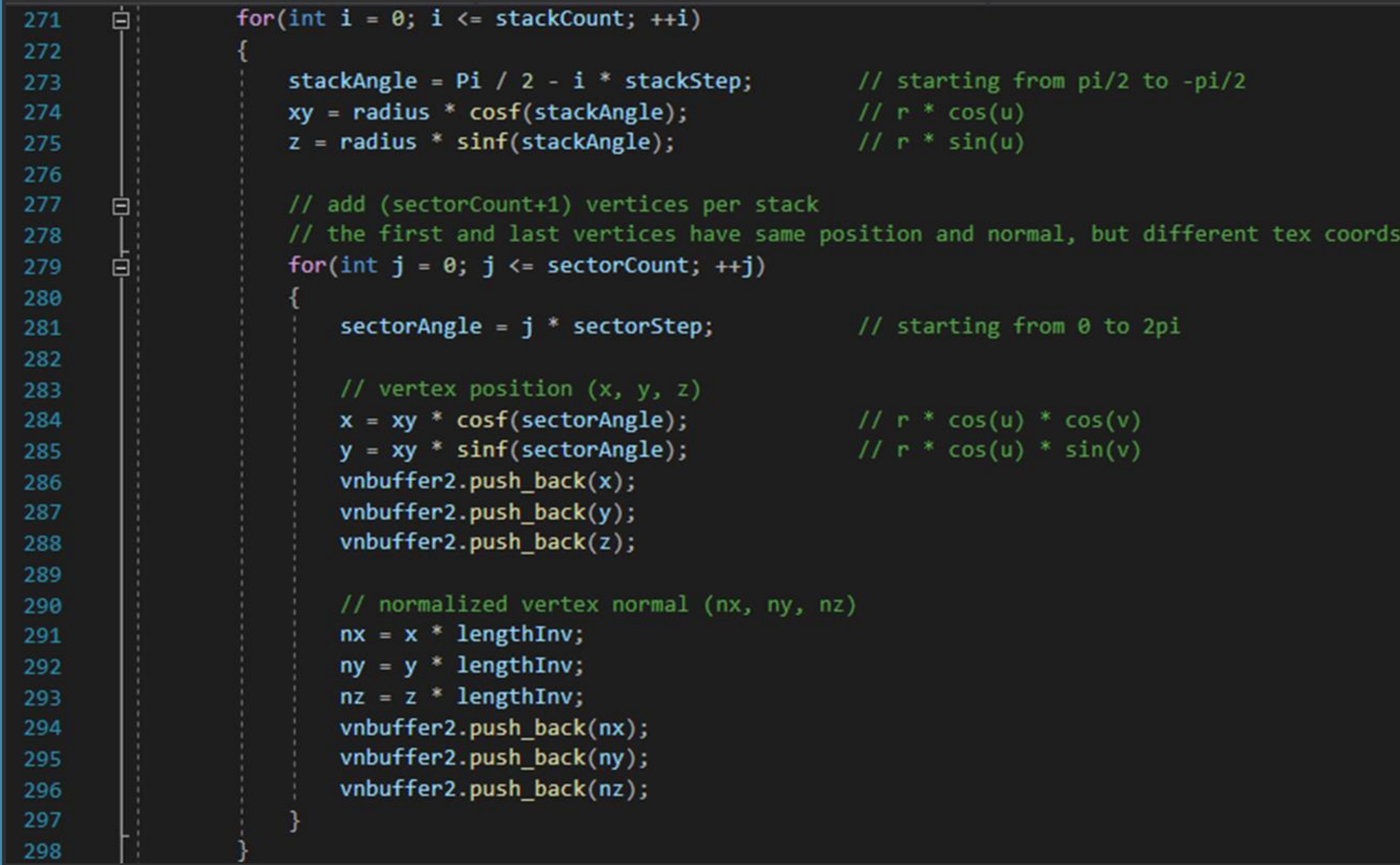
- range of sector angles is from 0 to 360 degrees
- stack angles are from 90 (top) to -90 degrees (bottom)
-

$$
\begin{aligned}
\theta &= 2\pi \cdot \frac{\text{sectorStep}}{\text{sectorCount}} \\
\phi &= \frac{\pi}{2} - \pi \cdot \frac{\text{stackStep}}{\text{stackCount}}
\end{aligned}
$$

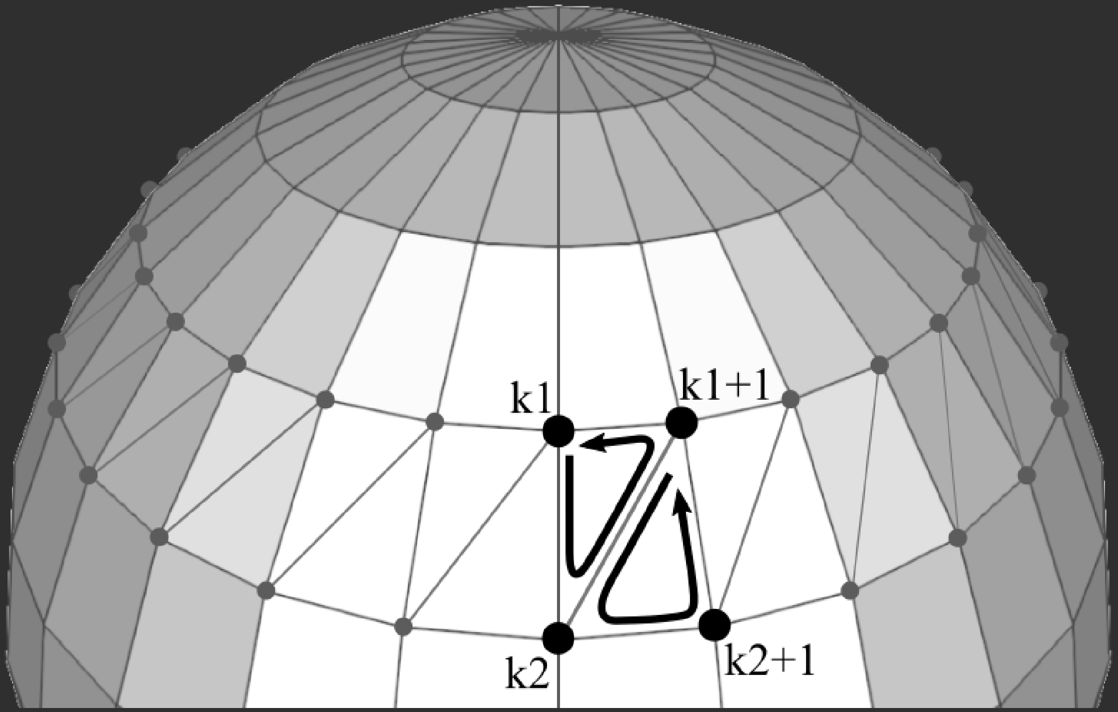
Creating the sphere

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

Creating the sphere: 1.step



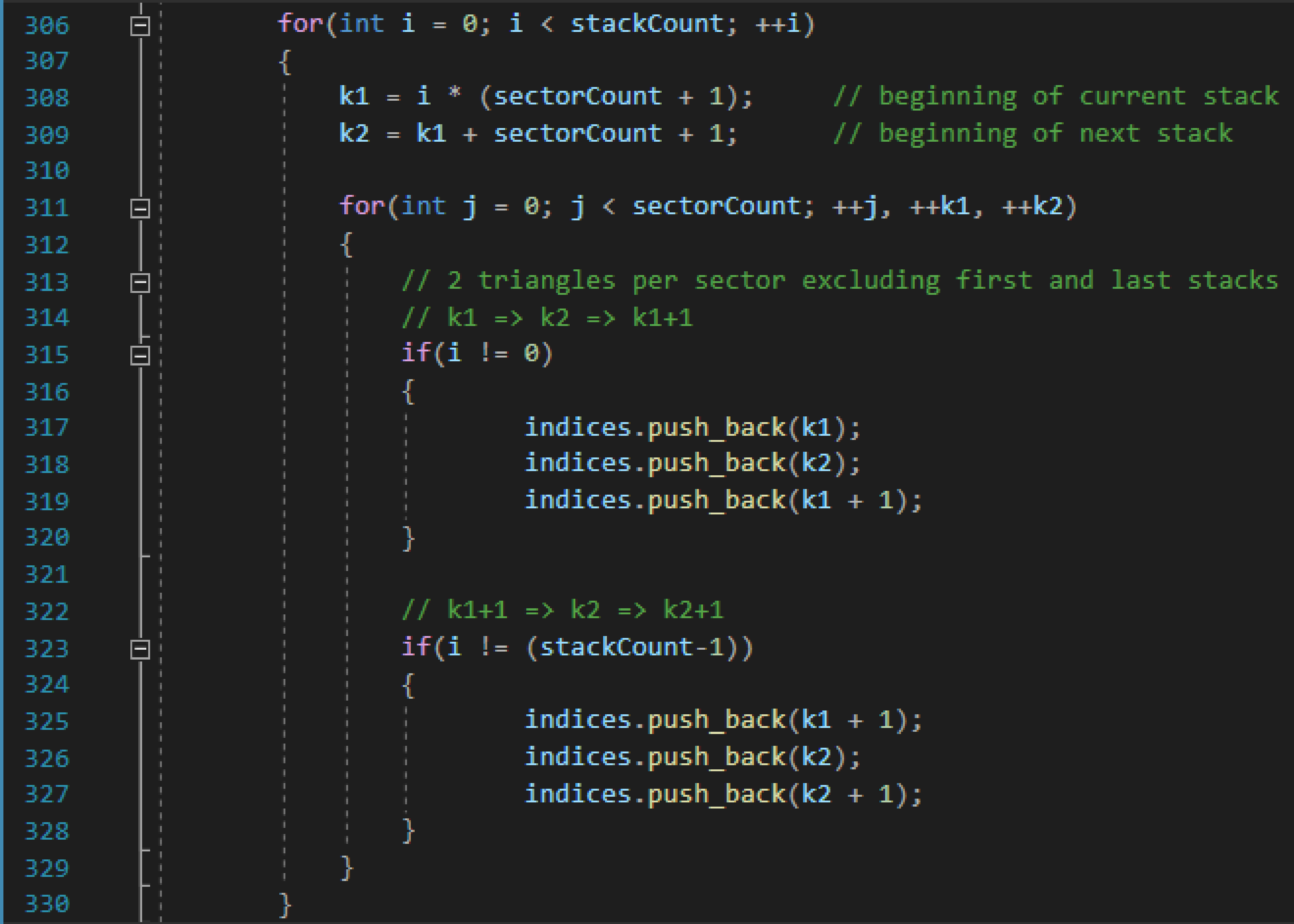
```
for(int i = 0; i <= stackCount; ++i)
{
    stackAngle = Pi / 2 - i * stackStep;        // starting from pi/2 to -pi/2
    xy = radius * cosf(stackAngle);             // r * cos(u)
    z = radius * sinf(stackAngle);              // r * sin(u)

    // add (sectorCount+1) vertices per stack
    // the first and last vertices have same position and normal, but different tex coords
    for(int j = 0; j <= sectorCount; ++j)
    {
        sectorAngle = j * sectorStep;           // starting from 0 to 2pi

        // vertex position (x, y, z)
        x = xy * cosf(sectorAngle);             // r * cos(u) * cos(v)
        y = xy * sinf(sectorAngle);             // r * cos(u) * sin(v)
        vnbuffer2.push_back(x);
        vnbuffer2.push_back(y);
        vnbuffer2.push_back(z);

        // normalized vertex normal (nx, ny, nz)
        nx = x * lengthInv;
        ny = y * lengthInv;
        nz = z * lengthInv;
        vnbuffer2.push_back(nx);
        vnbuffer2.push_back(ny);
        vnbuffer2.push_back(nz);
    }
}
```



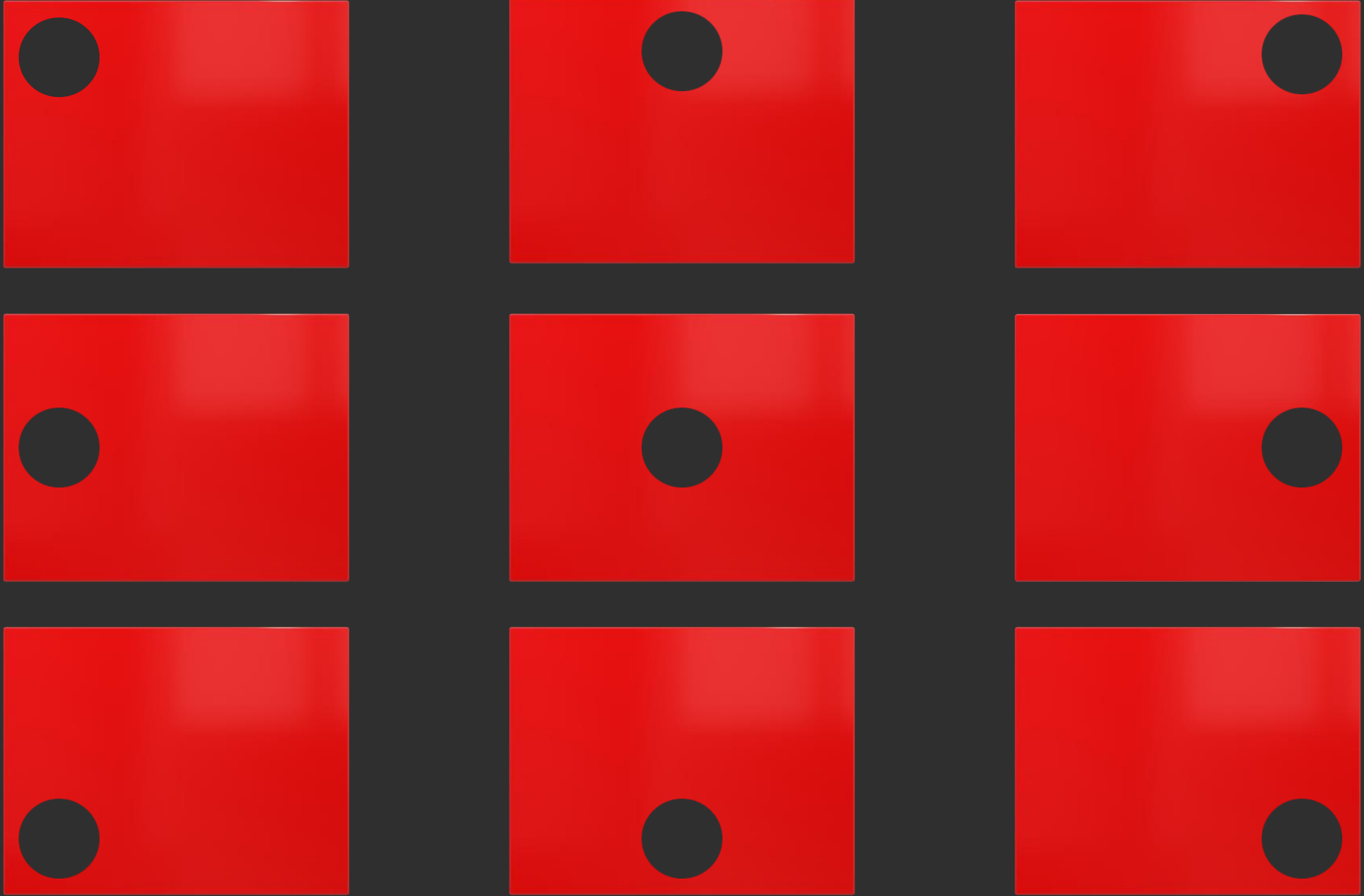
Sphere

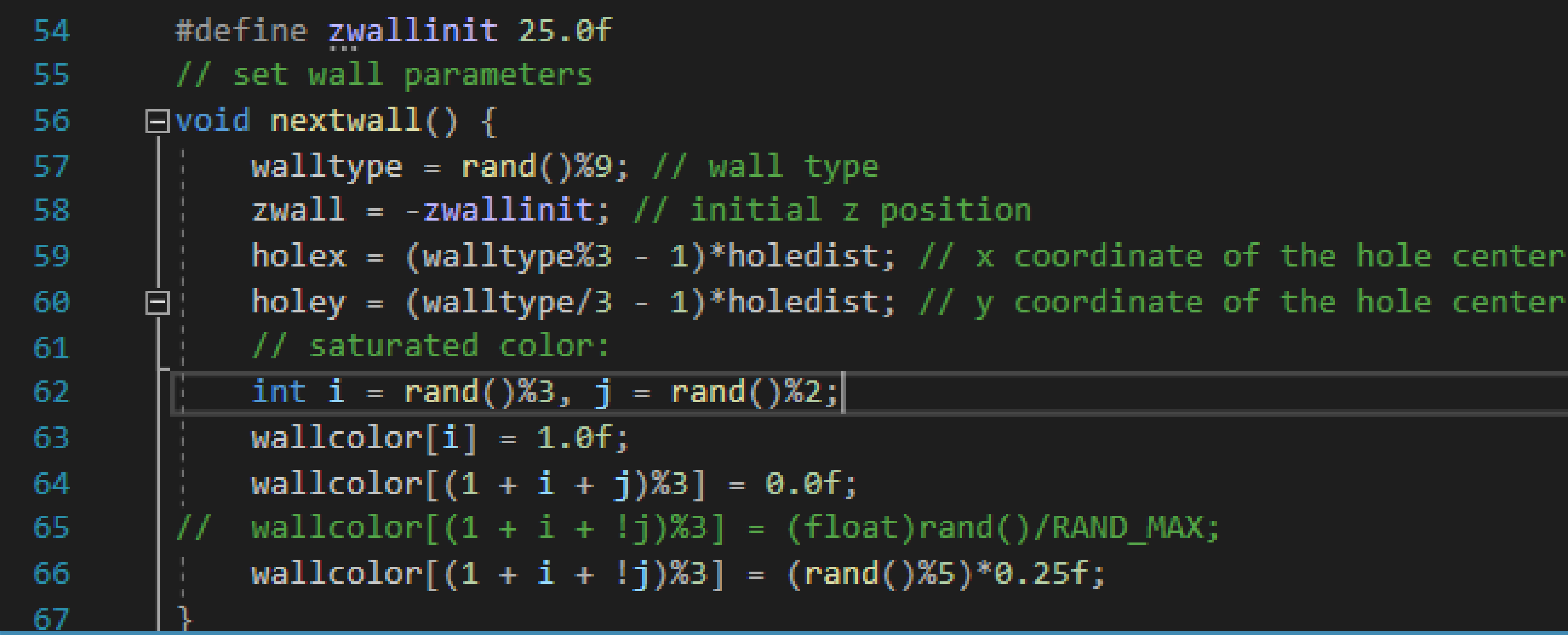
- to draw the surface of a sphere in OpenGL, you must triangulate adjacent vertices to form polygons
- Each sector in a stack requires 2 triangles
- If the first vertex index in the current stack is k1 and the next stack is k2, then the counterclockwise orders of vertex indices of 2 triangles are;
-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

Creating the sphere: 2.step



```
for(int i = 0; i < stackCount; ++i)
{
    k1 = i * (sectorCount + 1);     // beginning of current stack
    k2 = k1 + sectorCount + 1;      // beginning of next stack

    for(int j = 0; j < sectorCount; ++j, ++k1, ++k2)
    {
        // 2 triangles per sector excluding first and last stacks
        // k1 => k2 => k1+1
        if(i != 0)
        {
                indices.push_back(k1);
                indices.push_back(k2);
                indices.push_back(k1 + 1);
        }

        // k1+1 => k2 => k2+1
        if(i != (stackCount-1))
        {
                indices.push_back(k1 + 1);
                indices.push_back(k2);
                indices.push_back(k2 + 1);
        }
    }
}
```





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

Creating walls

Demo