Department of Computer Science

(Department of Computer Science)

Jakob-Haringer-Straße 2

5020 Salzburg, Austria

Introduction computer graphics

Ballwall

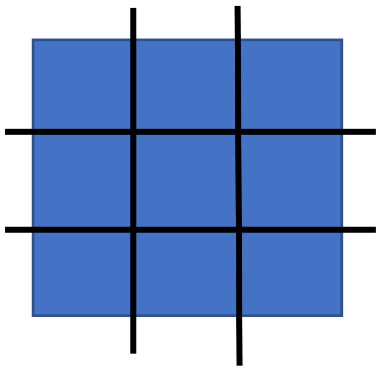
Samardzic Nuris, 01611855, nuris.samardzic@stud.sbg.ac.at

Tosun Begüm, 11719058, beguem.tosun@stud.sbg.ac.at

Idea

The basic idea of the ball wall is based on the concept of the 2D game "Flappy Bird". There is a sphere corresponding to the bird and walls with a hole each corresponding to the tubes of "Flappy Bird".

The walls move towards the ball and the player must move the ball with the keys "W" (up), "D" (to the right), "S"(down) and "A" (to the left) so that the ball passes through the hole and does not touch the wall.



The adjacent figure shows the nine possible positions where a hole can be located. The positioning options result in nine different walls.

Libraries

glew v2.2.0

The OpenGL Extension Wrangler Library (GLEW) is an open source cross-platform library for loading C/C++ extensions. GLEW provides efficient runtime mechanisms to determine which OpenGL extensions are supported on the target platform. The OpenGL core and extension functionality is provided in a single header file. GLEW has been tested on a variety of operating systems, including Windows, Linux, Mac OS X, FreeBSD, Irix, and Solaris.

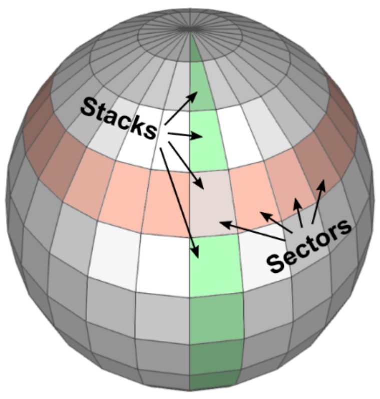
freeglut v3.0.0

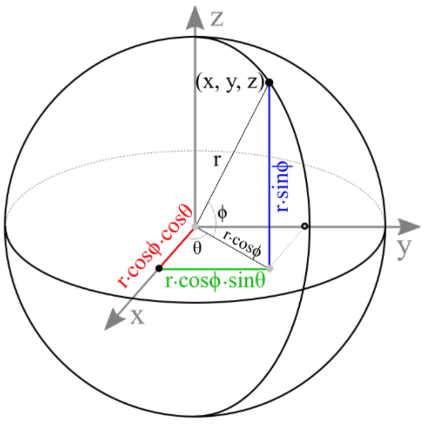
freeglut is a free software/open source alternative to the OpenGL Utility Toolkit (GLUT) library. GLUT was originally written by Mark Kilgard to support the sample programs in the second edition of OpenGL 'RedBook'. Since then, GLUT has been used in a variety of practical applications, as it is simple, widely used and very portable. The original GLUT library seems to have been abandoned with the latest version (3.7) from August 1998. Your license does not allow anyone to distribute modified library code.

sphere

The definition of a sphere is a closed 3D surface where each point on the sphere has the same distance (radius) from a particular point. The spherical equation: $x^2 + y^2 + z^2 = r^2$.

Since we cannot draw all the points on a sphere, we only capture a limited number of points by dividing the sphere by sectors (longitude) and stacks (latitude). Then connect these sampled points together to form surfaces of the sphere.





Any point (x, y, z) on a sphere can be calculated by parametric equations with the corresponding Sektor-Winkel θ and stack angle φ.

$$\begin{aligned} x &= (r \cdot \cos\phi) \cdot \cos\theta \\ y &= (r \cdot \cos\phi) \cdot \sin\theta \\ z &= r \cdot \sin\phi \end{aligned}$$

The range of sector angles ranges from 0 to 360 degrees and the stack angles range from 90 (top) to -90 degrees (bottom). The sector and stack angle for each step can be calculated as follows;

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

To draw the surface of a sphere in OpenGL, one must triangulate adjacent vertices to form polygons. It is possible to use a single triangle strip to render the entire sphere. However, if the common vertices have different normals or texture coordinates, a single triangular strip cannot be used.

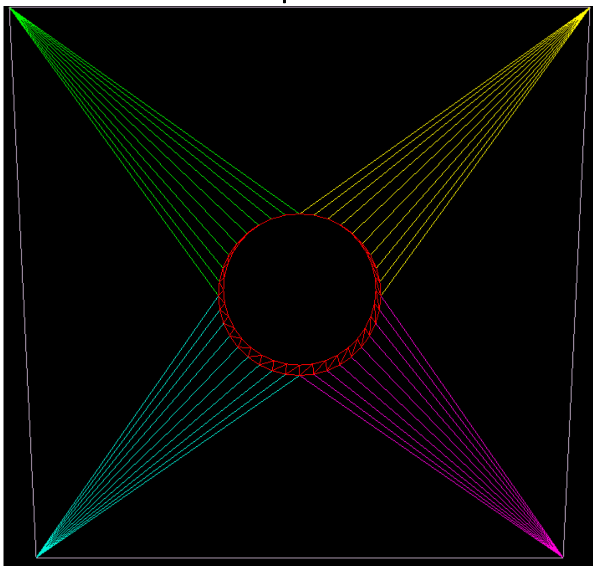
Each sector in a stack requires 2 triangles. If the first vertex index in the current stack is k1 and the next stack is k2, then the sequences of vertex indices of 2 triangles are counterclockwise.

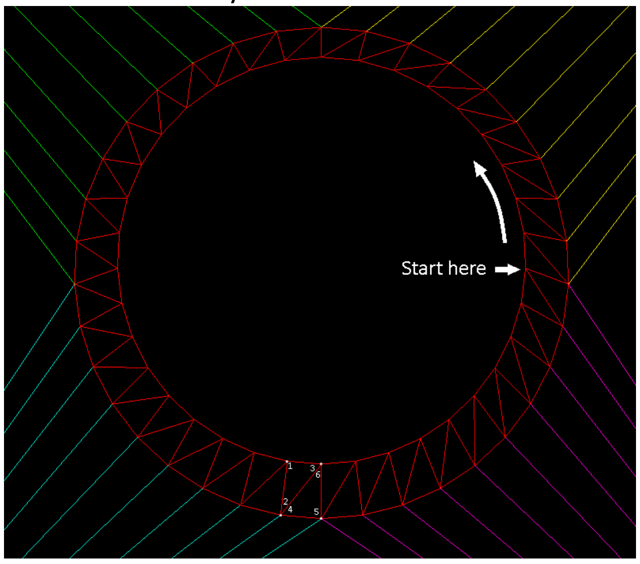
k1 ⟶ k2 ⟶ k1+1

k1+1 ⟶ k2 ⟶ k2+1

The creation of hole and wall

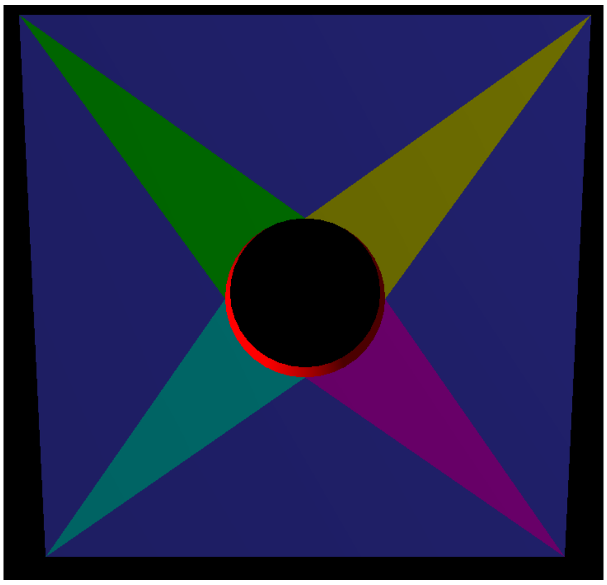
Before the wall is created, the hole must be created. First, the boundary of the hole is created. Then the triangles are created in the following figure. These are nothing more than the connection of the respective vertex of the wall with the boundary of the hole.

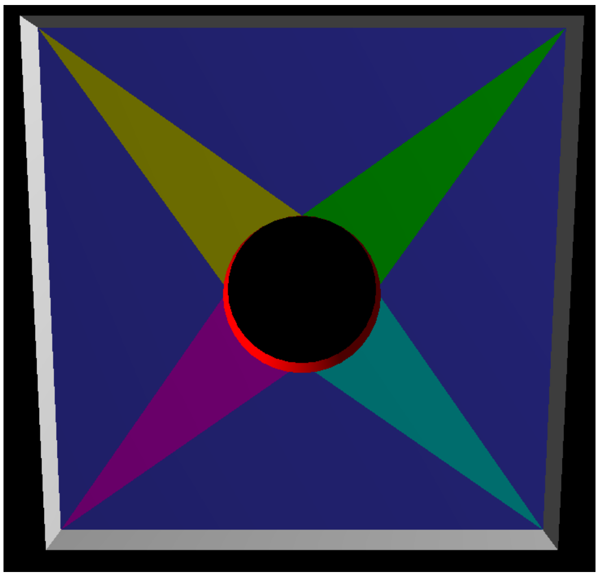




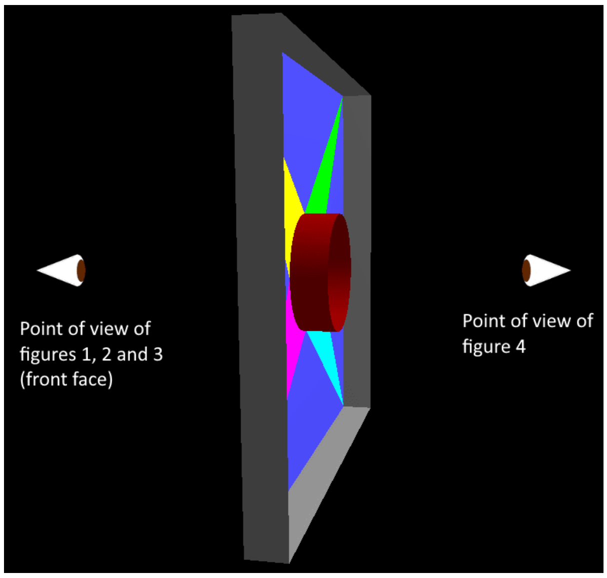
As can be seen in the figure on the right above, the creation of the "colorful triangles" is started with the yellowtriangle. With the order counterclockwise, the other triangles are then generated.

Thus, the creation of the hole is completed and we start with the creation of the wall. To do this, the blue triangles in the following figure on the left are first created. The grey sides of





the wall are then generated. As can be clearly seen in the next illustration, in reality it is not a wall at all. However, since the player will never see the "wall" from behind, it is also not necessary to create the rear side of the wall u. The next figure shows the created figure – "wall" – from behind.

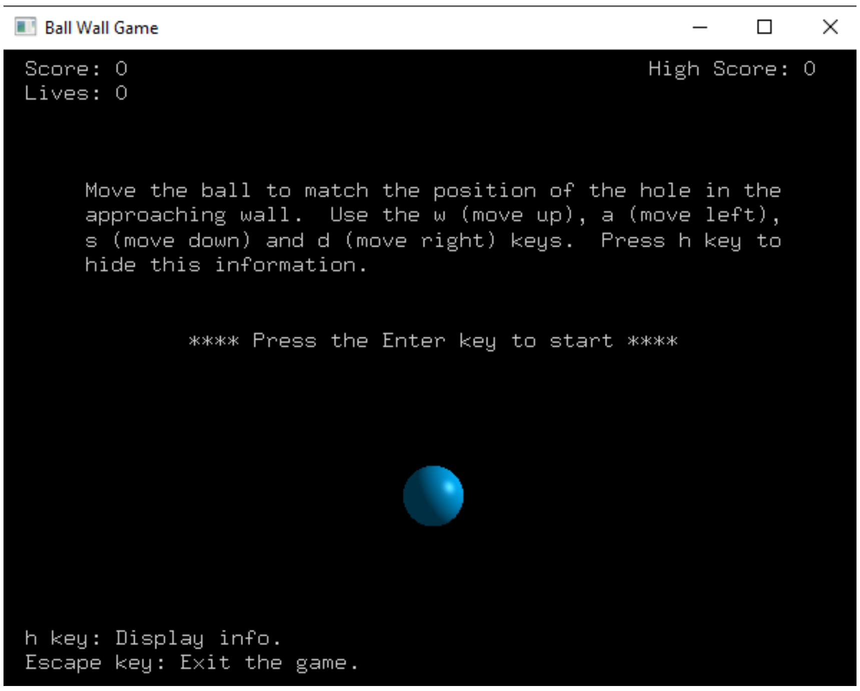


The hole and components of the wall were generated with the code on the next two pages.

```
struct v6f {
      float v[6];
      };

      float r = holeradius; // hole radius
      float h = holedist; // distance between hole centers
      int N = 36; // sides of circle, this value must be a multiple of 4.
      float d = h + 1.4f*r; // half side length
      float z = 0.09f*d; // half wall thickness
      glGenVertexArrays(9, wallId);
      // generate the vertex array indices for all 9 walls

      for (int j = 0; j != 9; j++) {
         vector<v6f> vnbuffer;
         {
            float cx = (j%3 - 1)*h; // hole center x coordinate
            float cy = (j/3 - 1)*h; // hole center y coordinate
            double angle = 0.0, delta = 2*M_PI/N;
            float c = 1.0, s = 0.0;
            for (int i = 0; i != N; i++) { // border of circular hole
               vnbuffer.push_back(v6f{r*c + cx, r*s + cy, -z, -c, -s, 0.0f});
               vnbuffer.push_back(v6f{r*c + cx, r*s + cy, z, -c, -s, 0.0f});
               angle += delta;
               s = (float)sin(angle); c = (float)cos(angle);
               vnbuffer.push_back(v6f{r*c + cx, r*s + cy, -z, -c, -s, 0.0f});
               vnbuffer.push_back(vnbuffer[vnbuffer.size() - 2]);
               vnbuffer.push_back(v6f{r*c + cx, r*s + cy, z, -c, -s, 0.0f});
               vnbuffer.push_back(v6f{r*c + cx, r*s + cy, -z, -c, -s, 0.0f});
            }

            angle = 0.0;
            c = 1.0; s = 0.0;
            for (int i = 0; i < N/4; i++) {
               // triangles from the upper right wall corner to the hole border
               vnbuffer.push_back(v6f{r*c + cx, r*s + cy, z, 0.0f, 0.0f, 1.0f});
               vnbuffer.push_back(v6f{d, d, z, 0.0f, 0.0f, 1.0f});
               angle += delta;
               s = (float)sin(angle); c = (float)cos(angle);
               vnbuffer.push_back(v6f{r*c + cx, r*s + cy, z, 0.0f, 0.0f, 1.0f});
            }

            angle = M_PI/2;
            c = 0.0; s = 1.0;
            for (int i = 0; i < N/4; i++) {
               // triangles from the upper left wall corner to the hole border
               vnbuffer.push_back(v6f{r*c + cx, r*s + cy, z, 0.0f, 0.0f, 1.0f});
               vnbuffer.push_back(v6f{-d, d, z, 0.0f, 0.0f, 1.0f});
               angle += delta;
               s = (float)sin(angle); c = (float)cos(angle);
               vnbuffer.push_back(v6f{r*c + cx, r*s + cy, z, 0.0f, 0.0f, 1.0f});
            }

            angle = M_PI;
            c = -1.0; s = 0.0;
            for (int i = 0; i < N/4; i++) {
               // triangles from the lower left wall corner to the hole border
               vnbuffer.push_back(v6f{r*c + cx, r*s + cy, z, 0.0f, 0.0f, 1.0f});
               vnbuffer.push_back(v6f{-d, -d, z, 0.0f, 0.0f, 1.0f});
               angle += delta;
               s = (float)sin(angle); c = (float)cos(angle);
               vnbuffer.push_back(v6f{r*c + cx, r*s + cy, z, 0.0f, 0.0f, 1.0f});
            }

            angle = 1.5*M_PI;
```

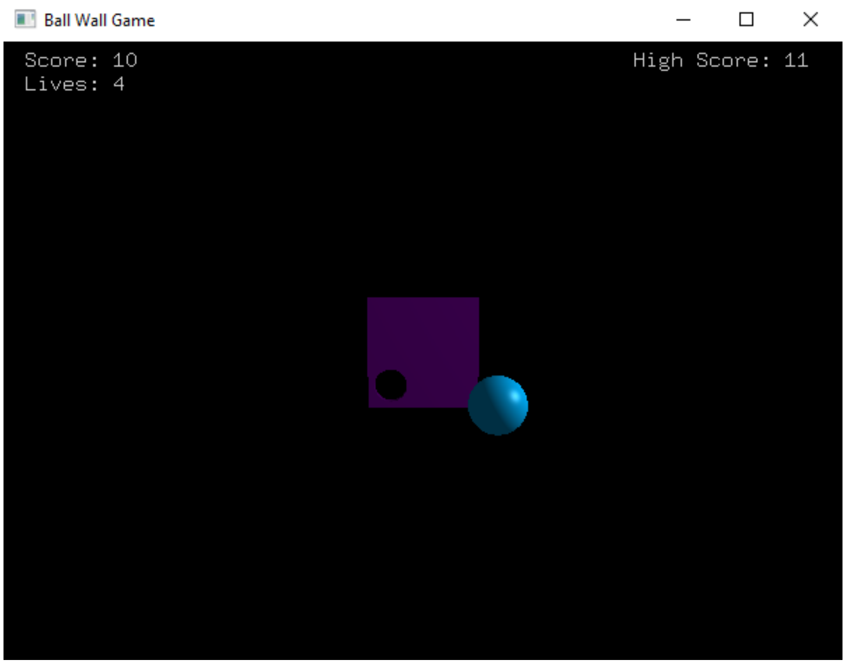
```
    c = 0.0; s = -1.0;
    for (int i = 0; i < N/4; i++) {
       // triangles from the lower right wall corner to the hole border
       vnbuffer.push_back(v6f{r*c + cx, r*s + cy, z, 0.0f, 0.0f, 1.0f});
       vnbuffer.push_back(v6f{d, -d, z, 0.0f, 0.0f, 1.0f});
       angle += delta;
       s = (float)sin(angle); c = (float)cos(angle);
       vnbuffer.push_back(v6f{r*c + cx, r*s + cy, z, 0.0f, 0.0f, 1.0f});
    }

    // missing wall front face triangles:
    vnbuffer.push_back(v6f{d, -d, z, 0.0f, 0.0f, 1.0f});
    vnbuffer.push_back(v6f{d, d, z, 0.0f, 0.0f, 1.0f});
    vnbuffer.push_back(v6f{r + cx, cy, z, 0.0f, 0.0f, 1.0f});
    vnbuffer.push_back(v6f{d, d, z, 0.0f, 0.0f, 1.0f});
    vnbuffer.push_back(v6f{-d, d, z, 0.0f, 0.0f, 1.0f});
    vnbuffer.push_back(v6f{cx, r + cy, z, 0.0f, 0.0f, 1.0f});
    vnbuffer.push_back(v6f{-d, d, z, 0.0f, 0.0f, 1.0f});
    vnbuffer.push_back(v6f{-d, -d, z, 0.0f, 0.0f, 1.0f});
    vnbuffer.push_back(v6f{cx - r, cy, z, 0.0f, 0.0f, 1.0f});
    vnbuffer.push_back(v6f{-d, -d, z, 0.0f, 0.0f, 1.0f});
    vnbuffer.push_back(v6f{d, -d, z, 0.0f, 0.0f, 1.0f});
    vnbuffer.push_back(v6f{cx, cy - r, z, 0.0f, 0.0f, 1.0f});
    // wall top side (+y)
    vnbuffer.push_back(v6f{d, d, z, 0.0f, 1.0f, 0.0f});
    vnbuffer.push_back(v6f{d, d, -z, 0.0f, 1.0f, 0.0f});
    vnbuffer.push_back(v6f{-d, d, -z, 0.0f, 1.0f, 0.0f});
    vnbuffer.push_back(v6f{-d, d, -z, 0.0f, 1.0f, 0.0f});
    vnbuffer.push_back(v6f{-d, d, z, 0.0f, 1.0f, 0.0f});
    vnbuffer.push_back(v6f{d, d, z, 0.0f, 1.0f, 0.0f});

    // wall bottom side (-y)
    vnbuffer.push_back(v6f{d, -d, -z, 0.0f, -1.0f, 0.0f});
    vnbuffer.push_back(v6f{d, -d, z, 0.0f, -1.0f, 0.0f});
    vnbuffer.push_back(v6f{-d, -d, -z, 0.0f, -1.0f, 0.0f});
    vnbuffer.push_back(v6f{-d, -d, z, 0.0f, -1.0f, 0.0f});
    vnbuffer.push_back(v6f{-d, -d, -z, 0.0f, -1.0f, 0.0f});
    vnbuffer.push_back(v6f{d, -d, z, 0.0f, -1.0f, 0.0f});

    // wall left side (-x)
    vnbuffer.push_back(v6f{-d, d, z, -1.0f, 0.0f, 0.0f});
    vnbuffer.push_back(v6f{-d, d, -z, -1.0f, 0.0f, 0.0f});
    vnbuffer.push_back(v6f{-d, -d, -z, -1.0f, 0.0f, 0.0f});
    vnbuffer.push_back(v6f{-d, -d, -z, -1.0f, 0.0f, 0.0f});
    vnbuffer.push_back(v6f{-d, -d, z, -1.0f, 0.0f, 0.0f});
    vnbuffer.push_back(v6f{-d, d, z, -1.0f, 0.0f, 0.0f});

    // wall right side (+x)
    vnbuffer.push_back(v6f{d, d, -z, 1.0f, 0.0f, 0.0f});
    vnbuffer.push_back(v6f{d, d, z, 1.0f, 0.0f, 0.0f});
    vnbuffer.push_back(v6f{d, -d, -z, 1.0f, 0.0f, 0.0f});
    vnbuffer.push_back(v6f{d, -d, z, 1.0f, 0.0f, 0.0f});
    vnbuffer.push_back(v6f{d, -d, -z, 1.0f, 0.0f, 0.0f});
    vnbuffer.push_back(v6f{d, d, z, 1.0f, 0.0f, 0.0f});
}
```

Spielaufbau



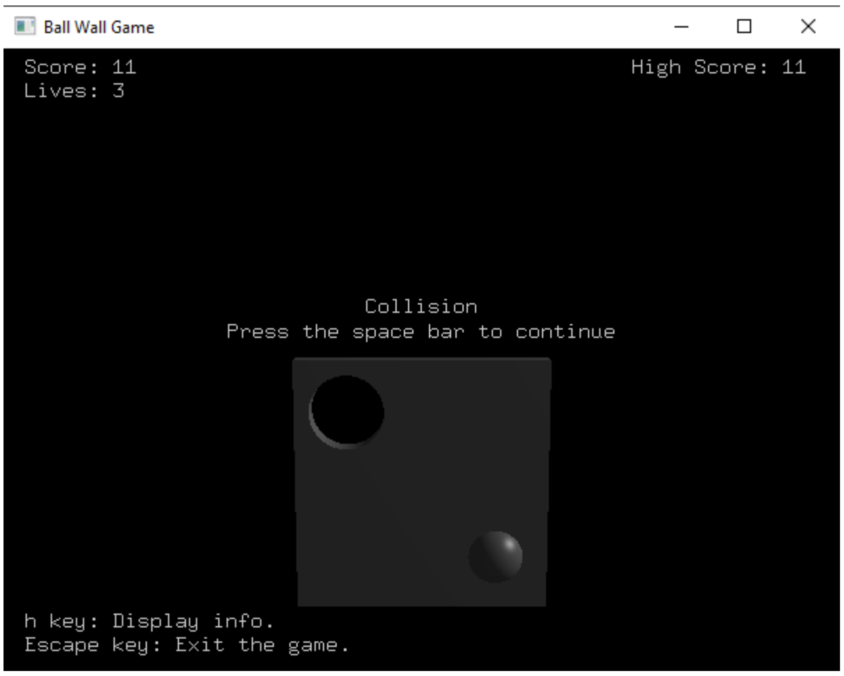
```
Score: 0                                                High Score: 0
Lives: 0

     Move the ball to match the position of the hole in the
     approaching wall.  Use the w (move up), a (move left),
     s (move down) and d (move right) keys.  Press h key to
     hide this information.

             **** Press the Enter key to start ****

h key: Display info.
Escape key: Exit the game.
```

In the start menu there are counters for score and lives on the upper left, a counter for the high score in the upper right corner, furthermore, the keys are used to play/end the game.



```
Score: 10                                              High Score: 11
Lives: 4
```

At the beginning, the player has three lives. After every tenth successfully traversed wall, the difficulty level is increased by one, i.e. the walls become faster and the player gets an extra life.



```
Score: 11                                   High Score: 11
Lives: 3

                  Collision
        Press the space bar to continue

h key: Display info.
Escape key: Exit the game.
```

If the ball touches a wall, the player loses a life. If the player has at least one more life, he can continue the game, otherwise he has to start anew.

Setting up the code

Files:

- ballwall.cpp (main code)
- loadCompileShaders.cpp (code for loading and compiling shaders)
- camera.cpp (class camera code)
- loadCompileShaders.h (header file)
- camera.h (header file)
- shader.vert (vertex shader code)
- shader.frag (fragment shader code)
- Makefile (MacOS g++ Makefile)
- makefileLinux.mak (Linux g++ Makefile)
- makefileMinGW-w64.mak (MSYS2 - MinGW-w64 Makefile)

Folders:

- vsproject (contains the MSVC solution and project files)
- glew-2.2.0 (glew v2.2.0 MSVC library for 32 and 64 bit)
- freeglut (freeglut v3.0.0 MSVC library for 32 and 64 bit)

COMPILATION INSTRUCTIONS:

Linux

- install freeglut and glew development packages in a terminal console
- move to the folder where the code is located
- type: make **-f makefileLinux.mak**

MSYS2 - MinGW-w64 (Windows)

- Open the msys64\msys2-console.
    - To install freeglut: **pacman -S mingw-w64-x86_64-freeglut**
    - To install glew: **pacman -S mingw-w64-x86_64-glew**
- close the msys2 console
- Open the msys64\mingw64-console with the command cd
- go to the folder where the code is located and type: **make -f makefileMinGW-w64.mak**

Visual Studio

- Open the *vsproject\ballwall.sln* file.
- Select Configuration Version and Platform x64.
- Compile the code.
- To run the program from the Visual Studio Console, first copy the *files shader.vert, shader.frag, freeglut\bin\x64\freeglut.dll* and *glew-2.2.0/bin/Release/x64/glew32.dll* to the project folder.
- To run the program through File Explorer, first copy the fileen, *freeglut\bin\x64\freeglut.dll* and *glew-2.2.0/bin/Release/x64/glew32.dll* to the folder where the executable file is located and run it binary file.

glew MSVC library downloaded from: https://github.com/nigels-com/glew Windows binaries for 32-bit and 64-bit

freeglut MSVC library downloaded from:
https://www.transmissionzero.co.uk/software/freeglut-devel/freeglut 3.0.0 MSVC package

source

http://www.songho.ca/opengl/gl_sphere.html (for Sphere and Shaders)