PARIS LODRON UNIVERSITÄT SALZBURG

Fachbereich Computerwissenschaften

(Department of Computer Science)

Jakob-Haringer-Straße 2

5020 Salzburg

Einführung Computergraphik

Ballwall

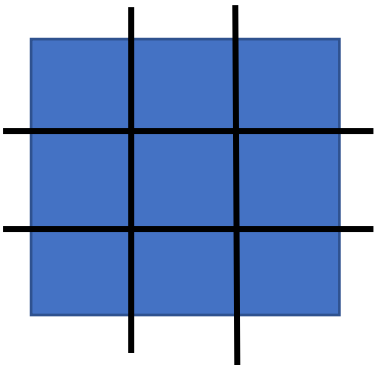
Samardzic Nuris, 01611855, nuris.samardzic@stud.sbg.ac.at

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

Grundidee

Die Grundidee vom Ballwall beruht auf das Konzept des 2D – Spiels „Flappy Bird“. Es gibt eine Kugel, die dem Vogel entspricht und Wände mit jeweils einem Loch, die den Röhren von „Flappy Bird“ entsprechen.

Die Wände bewegen sich auf den Ball hinzu und der Spieler muss die Kugel mit den Tasten „W“ (nach oben), „D“ (nach rechts), „S“ (nach unten) und „A“ (nach links) so bewegen, dass die Kugel durch das Loch durchgeht und nicht die Wand berührt.



Die nebenstehende Abbildung zeigt die neun möglichen Positionen, an denen sich ein Loch befinden kann. Durch die Positionierungsmöglichkeiten ergeben sich neun verschiedene Wände.

Libraries

glew v2.2.0

Die OpenGL Extension Wrangler Library (GLEW) ist eine plattformübergreifende Open-Source-Bibliothek zum Laden von C/C++-Erweiterungen. GLEW bietet effiziente Laufzeitmechanismen, um zu bestimmen, welche OpenGL-Erweiterungen auf der Zielplattform unterstützt werden. Die OpenGL-Kern- und Erweiterungsfunktionalität wird in einer einzigen Header-Datei bereitgestellt. GLEW wurde auf einer Vielzahl von Betriebssystemen getestet, darunter Windows, Linux, Mac OS X, FreeBSD, Irix und Solaris.

freeglut v3.0.0

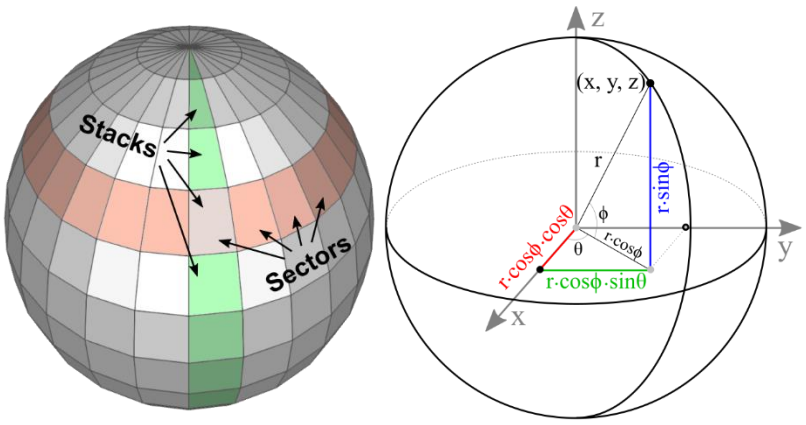
freeglut ist eine kostenlose Software/Open-Source-Alternative zur OpenGL Utility Toolkit (GLUT)-Bibliothek. GLUT wurde ursprünglich von Mark Kilgard geschrieben, um die Beispielprogramme in der zweiten Ausgabe von OpenGL 'RedBook' zu unterstützen. Seitdem wird GLUT in einer Vielzahl praktischer Anwendungen eingesetzt, da es einfach, weit verbreitet und sehr portabel ist. Die ursprüngliche GLUT-Bibliothek scheint mit der neuesten

Version (3.7) vom August 1998 aufgegeben worden zu sein. Ihre Lizenz erlaubt es niemandem, modifizierten Bibliothekscode zu verteilen.

Kugel

Die Definition einer Kugel ist eine geschlossene 3D-Oberfläche, bei der jeder Punkt auf der Kugel den gleichen Abstand (Radius) von einem bestimmten Punkt hat. Die Kugelgleichung: $x^2 + y^2 + z^2 = r^2$.

Da wir nicht alle Punkte auf einer Kugel zeichnen können, erfassen wir nur eine begrenzte Anzahl von Punkten, indem wir die Kugel durch Sektoren (Längengrad) und Stapel (Breitengrad) teilen. Verbinden Sie dann diese abgetasteten Punkte miteinander, um Oberflächen der Kugel zu bilden.



Ein beliebiger Punkt (x, y, z) auf einer Kugel kann durch parametrische Gleichungen mit dem entsprechenden Sektorwinkel θ und Stapelwinkel ϕ berechnet werden.

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

Der Bereich der Sektorwinkel reicht von 0 bis 360 Grad und die Stapelwinkel reichen von 90 (oben) bis -90 Grad (unten). Der Sektor- und Stapelwinkel für jeden Schritt kann wie folgt

berechnet werden;

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

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

Um die Oberfläche einer Kugel in OpenGL zu zeichnen, muss man benachbarte Scheitelpunkte triangulieren, um Polygone zu bilden. Es ist möglich, einen einzelnen Dreiecksstreifen zu verwenden, um die gesamte Kugel zu rendern. Wenn die gemeinsamen Scheitelpunkte jedoch unterschiedliche Normalen oder Texturkoordinaten aufweisen, kann kein einzelner Dreiecksstreifen verwendet werden.

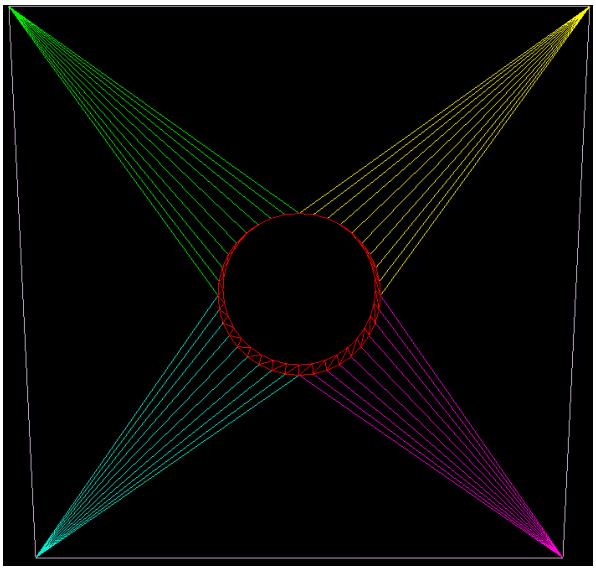
Jeder Sektor in einem Stapel erfordert 2 Dreiecke. Wenn der erste Scheitelpunktindex im aktuellen Stapel k1 ist und der nächste Stapel k2 ist, dann sind die Reihenfolgen der Scheitelpunktindizes von 2 Dreiecken gegen den Uhrzeigersinn;

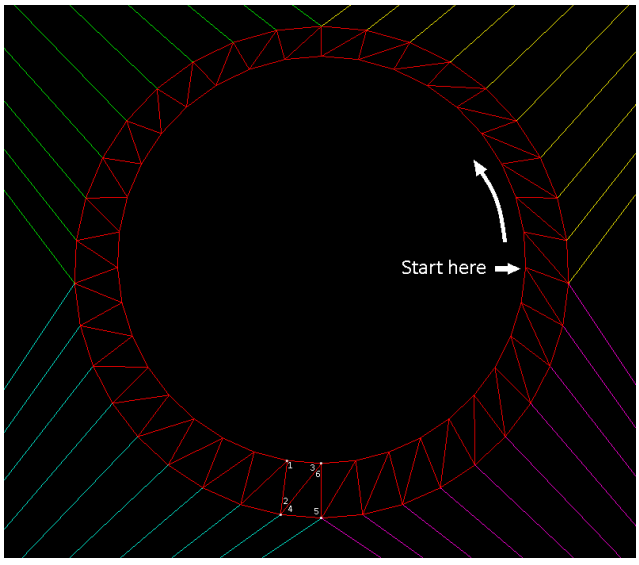
k1 ⟶ k2 ⟶ k1+1

k1+1 ⟶ k2 ⟶ k2+1

Die Erstellung von Loch und Wand

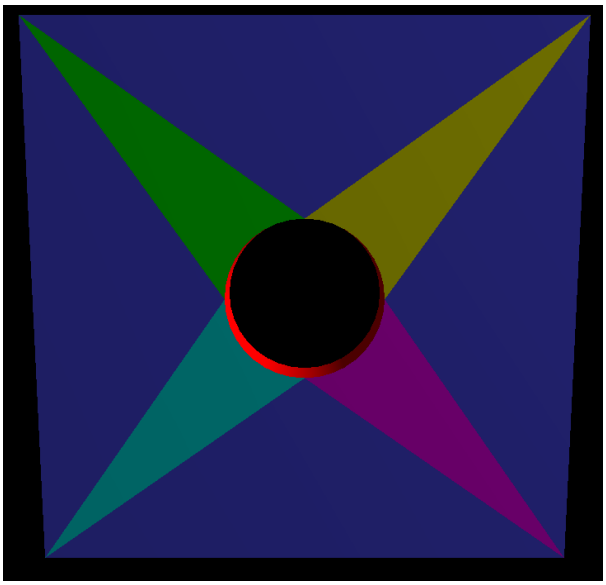
Bevor die Wand erstellt wird, muss das Loch erstellt werden. Zuerst wird die Grenze des Lochs erstellt. Dann werden die Dreiecke in der folgenden Abbildung erstellt. Diese sind nichts anderes als die Verbindung des jeweiligen Eckpunkts der Wand mit der Grenze des Lochs.

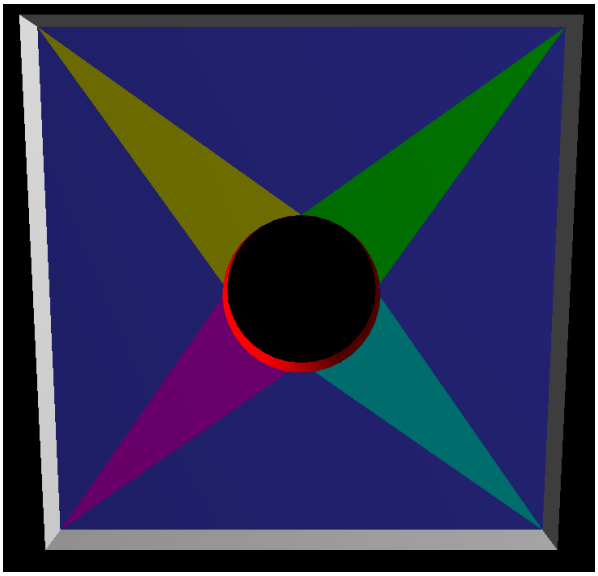




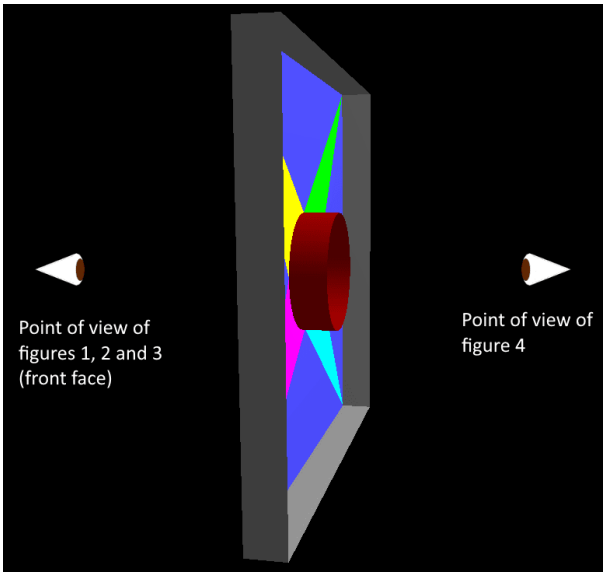
Wie in der obigen Abbildung rechts zu sehen ist, wird bei der Erstellung der „bunten Dreiecke“ bei dem gelben Dreieck begonnen. Mit der Reihenfolge gegen den Uhrzeigersinn werden dann die weiteren Dreiecke generiert.

Somit ist die Erstellung des Lochs abgeschlossen und wir beginnen mit der Erstellung der Wand. Dazu werden zuerst die blauen Dreiecke in der folgenden Abbildung links erstellt.





Anschließend werden die grauen Seiten der Wand generiert. Wie in der nächsten Abbildung klar zu sehen ist, handelt es sich in Wirklichkeit gar nicht um eine Wand. Da der Spieler aber die „Wand“ nie von hinten sehen wird, ist es auch nicht nötig die hintere Seite der Wand u erstellen. Die nächste Abbildung zeigt die erstellte Figur – „Wand“- von hinten.

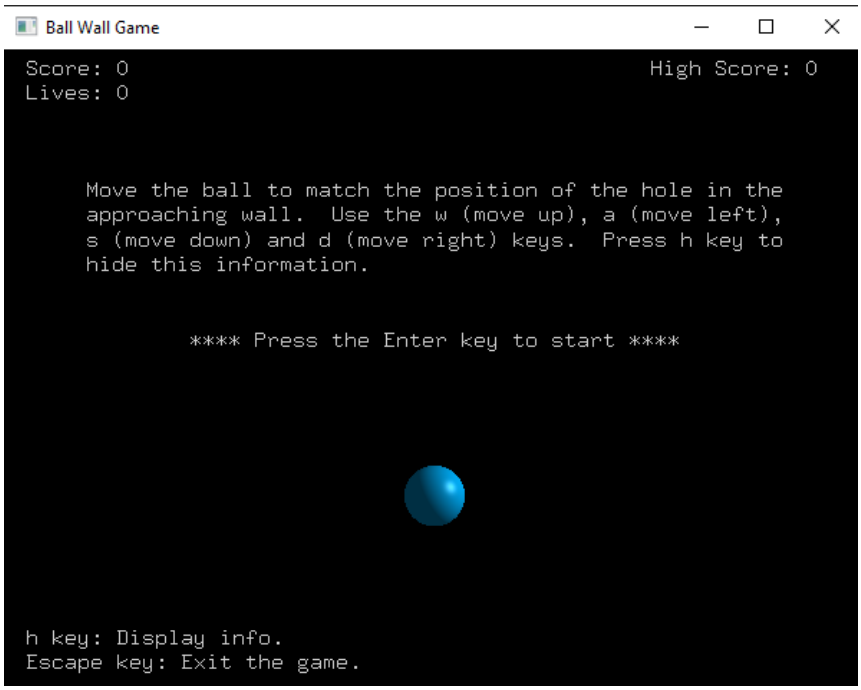


Das Loch und die Bestandteile der Wand wurden mit dem Code auf den nächsten zwei Seiten generiert.

```
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;
```

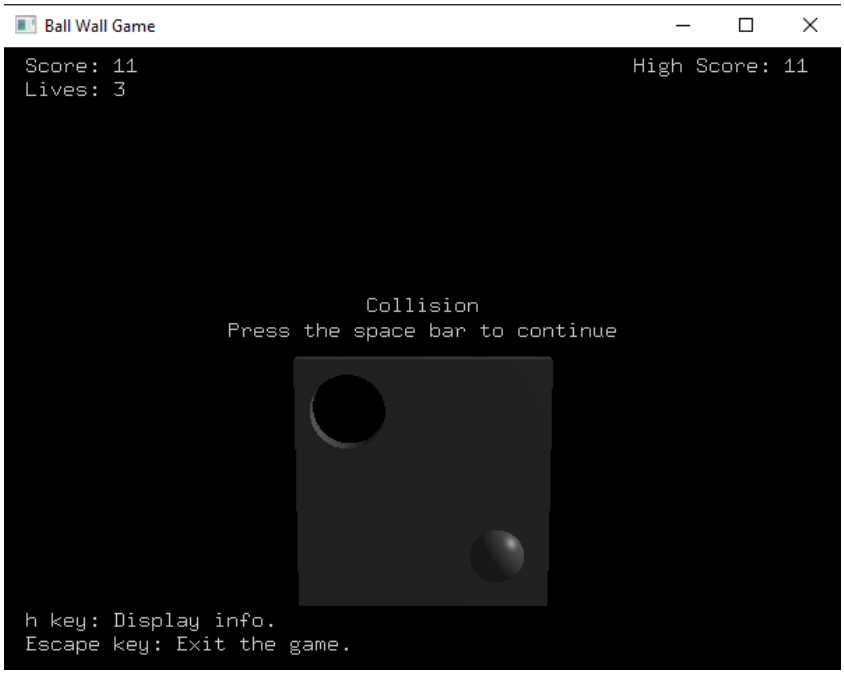
```
    c = 0.0; s = -1.0;
    for (int i = 0; i < N/4; i++) {
       // triangles from the lower right wall corner to the hole border
       vnbuffer.push_back(v6f{r*c + cx, r*s + cy, z, 0.0f, 0.0f, 1.0f});
       vnbuffer.push_back(v6f{d, -d, z, 0.0f, 0.0f, 1.0f});
       angle += delta;
       s =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



Im Startmenu befinden sich links oben Zähler für Score und Lives, rechts oben ein Zähler für den High Score, weiters werden noch angegeben mit welchen Tasten das Spiel gespielt/beendet wird.

Zu Beginn hat der Spieler drei Leben. Nach jeder zehnten erfolgreich durchquerten Wand wird die Schwierigkeitsstufe um eines erhöht, d.h. die Wände werden schneller und der Spieler bekommt ein zusätzliches Leben.



Berührt die Kugel eine Wand, verliert der Spieler ein Leben. Hat der Spieler mindestens noch ein Leben kann er das Spiel fortsetzten, ansonsten muss er neu anfangen.

Aufsetzung vom Code

Files: ballwall.cpp (main code) loadCompileShaders.cpp (Code zum Laden und Kompilieren von Shadern) 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 (enthält die MSVC-Lösung und Projektdateien) glew-2.2.0 (glew v2.2.0 MSVC-Bibliothek für 32 und 64 Bit) freeglut (freeglut v3.0.0 MSVC-Bibliothek für 32 und 64 Bit)

Um in MacOS zu kompilieren (nicht getestet):

Wechseln Sie in einer Terminalkonsole in den Ordner, in dem sich der Code befindet, und geben Sie Folgendes ein: make

Unter Linux kompilieren (nicht getestet):

Installieren Sie Freeglut- und Glew-Entwicklungspakete in einer Terminalkonsole, wechseln Sie in den Ordner, in dem sich der Code befindet, und geben Sie Folgendes ein:
make -f makefileLinux.mak

Kompilieren mit MSYS2 - MinGW-w64 (Windows):

Öffnen Sie die msys64\msys2-console. Um freeglut zu installieren:
pacman -S mingw-w64-x86_64-freeglut

Um glew zu installieren: pacman -S mingw-w64-x86_64-glew
schließen Sie die msys2-Konsole. Öffnen Sie die msys64\mingw64-console mit dem Befehl cd, wechseln Sie in den Ordner, in dem sich der Code befindet, und geben Sie Folgendes ein:
make -f makefileMinGW-w64.mak

Kompilieren mit Visual Studio Community 2019:

Öffnen Sie die Datei vsproject\ballwall.sln . Wählen Sie Konfigurationsversion und Plattform x64. Kompilieren Sie den Code. Um das Programm über die Visual Studio Console auszuführen, kopieren Sie zunächst die Dateien shader.vert, shader.frag und freeglut\bin\x64\freeglut.dll in den Ordner vsproject. Um das Programm über den Datei-

Explorer auszuführen, kopieren Sie zunächst die Datei freeglut\bin\x64\freeglut.dll in den Ordner, in dem sich die ausführbare Datei befindet.

glew MSVC-Bibliothek heruntergeladen von: https://github.com/nigels-com/glew Windows-Binärdateien für 32-Bit und 64-Bit

freeglut MSVC-Bibliothek heruntergeladen von:
https://www.transmissionzero.co.uk/software/freeglut-devel/freeglut 3.0.0 MSVC-Paket

Quelle

http://www.songho.ca/opengl/gl_sphere.html (für Kugel und Shaders)