BloXphere

Johannes Brunner – 1025064 – 033532

Michael Urbanek – 1328186 – 066935

Tastaturbelegung

- R – Aktiviert / Deaktiviert das Partikelsystem für die Flammenspur
- V – Viewchange (Überfahren eines roten Blockes) erzwingen
- P – Aktiviert / Deaktiviert Physics
- SPACE – Startet ein neues Spiel [FORCE NEW GAME]
- ENTER – Startet ein neues Spiel nach Game Over
- M – Aktiviert / Deaktiviert die Musik inkl. Beat Detection
- Page UP / Down erhöht bzw. reduziert den Gamma-Value
- B – Aktiviert / Deaktiviert die Bewegung des schwarzen Lochs

Um das Spiel zu Testen und sich in Ruhe umzusehen, geht bitte wie folgt vor:

Deaktiviert mittels B das **schwarze Loch**, wenn euch die **Musik** nervt mittels M die Musik, wenn euch der Bildschirm zu dunkel ist mittels Page UP / Down den **Gammawert**. Wenn ihr euch **umsehen** wollt, so nutzt V für einen Viewchange.

Solltet ihr in den Code müssen, so befindet sich die .sln Datei auf GIT. In

- BloXphere/BloXphere [befinden sich die .h und .cpp sowie Unterordner mit den lvls, Shadern, Textures, usw.)
- BloXphere/x64 der Debug/Release Ordner [wo die .exe reingebuilded wird]
- BloXphere/BloXphere/x64 [ein Ordner von Visual Studio, der nur .obj Dateien enthält -> unwichtig]
- BloXphere/extern [die include Files der externen Libraries)
- BloXphere/lib [die Libraries].

Start der .exe mittels Übergabeparameter

D:\Cplusplus\OpenGLTest\BloXphere\x64\bin\BloXphere.exe 1024 768 full 60 AnTx

Die ersten beiden Parameter definieren die Größe des Fensters. Der dritte Parameter kann „full“ (Vollbild) oder „no“ sein (nicht Vollbild). Der vierte Parameter bestimmt die Wiederholungsrate z.B. 60 Hz. Der letzte Parameter den Spielernamen.

Kriterien

Effects

Burning Way (GPU-Particle System (+Transform Feedback,Instancing)) [1.5 Punkte]

Die Flammenspur, der Regenbogenschweif des Ponys wie auch der Sternenschweif den die Blöcke die um das schwarze Loch kreisen zurücklassen wurden mittels eines Particle Systems implementiert. Dabei wird für jedes implementierte Particle System beim Initialisieren ein Seedpartikel gesetzt und in einen Buffer geschrieben welcher der GPU übergeben wird. Von diesem Seedpartikel ausgehend werden dann neue Partikel in der GPU erzeugt und in den Feedbackbuffer geschrieben. Auch sämtliche Transformtionen bzw. Partikelbewegungen für jedes Partikel werden auf der GPU pro Frame berechnet und die Partikel mit den neuen Partikeln in den Feedbackbuffer geschrieben und danach gerendert. Alle Informationen welche die GPU zum erzeugen oder bearbeiten der Partikel braucht sind entweder im Seedpartikel vorhanden oder werden mittels uniform übergeben.

Mirrors (Render-To-Texture, Framebuffers) [1.0 Punkte]

Die Mirrors wurden mittels Render-To-Texture und Framebuffers umgesetzt. Die Szene wird pro Mirror aus einer anderen Perspektive neu gerendert und auf die Texture wiedergegeben. Die Kamera wurde so positioniert, dass sie direkt hinter dem Spiegel sitzt sodass der Spiegel auch wie ein Spiegel wirkt ;-). Eine Instanz der Klasse Mirror entspricht einem Spiegel. In der Magician.cpp wird in den jeweiligen Framebuffer geschrieben.

Post-Image-Processing Whirl Effect (Post-Processing-Shader) [1.0 Punkte]

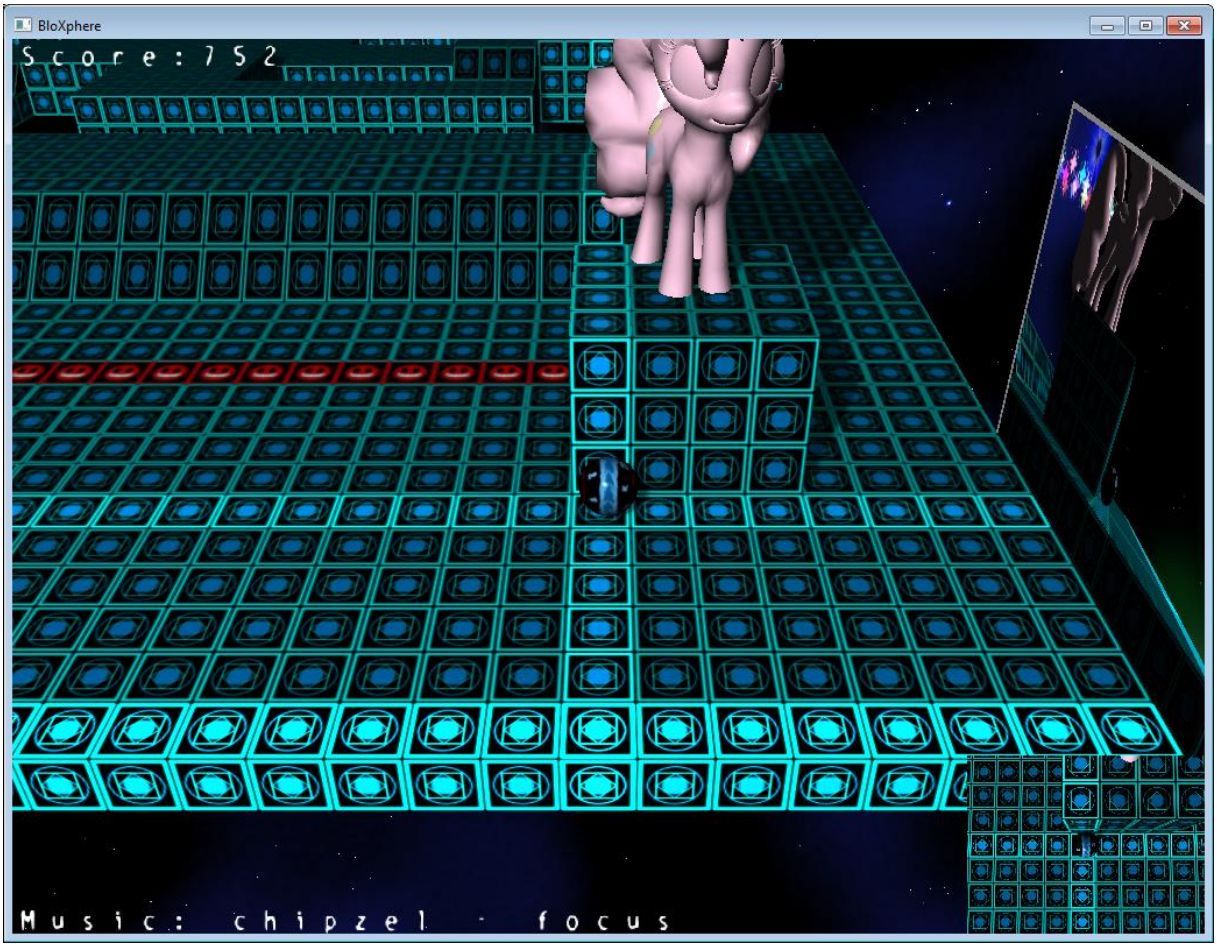
Der Whirl Effekt (also ein Verzerren des Bildschirms, sobald das schwarze Loch zu Nahe kommt) wurde durch einen Post-Processing-Shader realisiert. Im Vertexshader werden die Koordinaten zwischen 0 und 1 normiert (um im Fragmentshader ein einfacheres Mapping zu ermöglichen). Im Fragmentshader werden dann die jeweiligen Pixel um den Winkel Theta rotiert, sodass der Whirleffect entsteht. Die Auflösung, der Radius sowie der ursprüngliche Winkel werden mittels uniform übergeben.

Shadow Maps (Shadow Maps (with PCF)) [1.5 Punkte]

Für das Shadow Mapping haben wir uns für die PCF Variante entschieden. Zuerst rendern wir die Szene ein weiteres Mal von der Lichtquelle in den Ursprung gehend (eigene Cam mit Parametern an der Position des Lichts). Jene Elemente, die Schatten empfangen können, erhalten die Shadowmap als Übergabeparamter in den jeweiligen Shader (Instancing-Shader, InstancingHalfcubes-Shader). In den beiden Fragmentshadern wird sodann entschieden, ob ein Fragment im Schatten liegt und wenn ja, wird es um einen Schattenfaktor dunkler gezeichnet. Da dadurch die Kanten recht hart wirken, sorgt ein 3x3 Filter an den Rändern für einen weicheren Übergang.

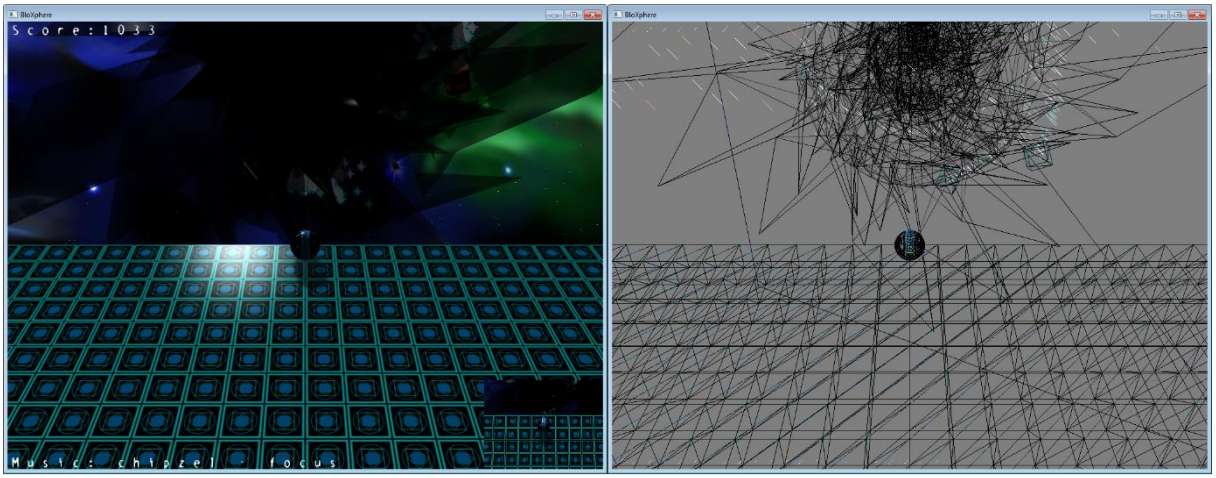
Complex Objects

Da ein komplexes Objekt dem Spielkonzept BloXphere widerspricht, haben wir uns dafür entschieden, ein rosafarbenes Einhorn zu Beginn des Spiels zu platzieren. Es dreht sich im Kreis (kann also gut beobachtet werden hinsichtlich Schattierung) und versprüht Glitzersterne. Zur Schattierung wird im Allgemein das Beleuchtungsmodell von Phong verwendet.



Animated Objects

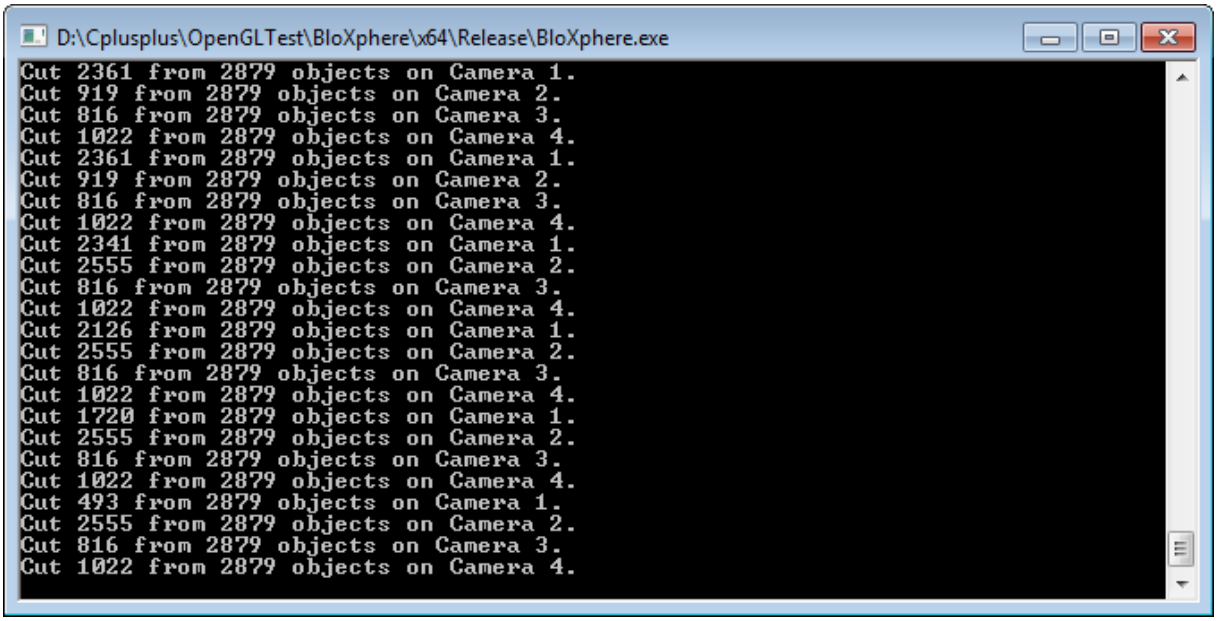
Als Animated Objects wurde von uns das schwarze Loch implementiert. Das schwarze Loch selbst teilt sich in mehrere Bereiche, die hierarchisch voneinander abhängen (die Objekte des schwarzen Loches sowie die umherfliegenden Würfel orientieren sich am BlackholeController). Die einzelnen Elemente werden zusätzlich zur Laufzeit skaliert, sodass ein Effekt des „Wabbelns“ entsteht. Zusätzlich hierzu bewegen sich Blöcke um das schwarze Loch herum, die wiederrum in Abhängigkeit zur relativen Position des Objekts stehen.



View-Frustum-Culling

Das View-Frustum-Culling wurde mittels Anleitung von lighthouse3D.com implementiert und überprüft mittels Bounding-Spheres welche Elemente in der Sichtweite liegen und welche nicht. Im

WorldGenerator.cpp befindet sich eine Funktion ViewFrustumCulling, welche sich um den jeweiligen Test sowie die richtige Weitergabe an die jeweiligen Objekte kümmert. Die eigentliche View-Frustum-Methodik wurde in der Camera.cpp File implementiert. CameraCalculateNewMVP() wird bei jeder Kameraänderung aufgerufen, sodass das View-Frustum automatisch angepasst wird. Da wir mit mehreren Kameras die Szene rendern, werden bei den Spiegeln nur jene Teile gerendert, die die „Spiegelkamera“ auch wirklich sieht, sodass hier Performanceoptimierungen stattfinden. Mittels F8 kann das View-Frustum-Culling deaktiviert werden.



Transparency

Transparency wird bei uns für das schwarze Loch und für sämtliche ParticleSysteme verwendet. Hierfür muss der Fragmentshader eine 4d Farbvektor zurückliefern, wobei der 4te wert hierbei dem Alphawert entspricht welcher die Transparency festlegt. Vor dem rendern muss hierfür das Alphablending mittel glEnable(GL_BLEND) enabled werden. Um bei den Partikeln zu gewährleisten das auch Partikel gezeichnet werden welche hinter anderen Partikeln liegen aber wegen der Transparency gesehen werden sollten muss der Tiefentest vor dem rendern des Particlesystem deaktiviert und danach wieder aktiviert werden. Mittels F9 kann die Transparency de- und aktiviert werden.

Experiment with OpenGL

Wir verwenden durchgehend Vertex-Buffer-Objects, Vertex Array Objects, Frame Buffer Objects. Mip Mapping sowie Textur-Sampling-Quality können mit der jeweiligen Funktionstaste (F4/F5) getoggelt werden. Die jeweiligen Veränderungen werden mittels Konsole ausgegeben. F2 aktiviert die FPS Anzeige, F3 den Wireframe Modus, F8 das Viewfrustum-Culling und F9 die Transparency.

Special Features

Beat Detection

Das Spiel bietet vollen Support für Beat Detection. Mittels Beatschlag beginnt das Level zu pulsieren. In der Soundklasse können die einzelnen Parameter für den Song eingestellt werden (wie viele Bins erzeugt werden beziehungsweise ab welchem Ausschlag man einen Ausschlag erkenntlich machen möchte).

Level Editor

Das Spiel bietet die Möglichkeit, jedem Spieler und jeder Spielerin ein eigenes Level zu kreieren. Im Editor kann man die bereits bestehenden Levelfiles abändern [so braucht man den Code nicht verändern] und nach eigenen Wünschen anpassen. Es muss in jedem Level zumindest ein Block, eine Rampe und ein Button vorkommen. Die Höhe im wird durch eine Ziffer zwischen 0 und 9 im File festgelegt. Mittels V kann man auf der Höhe 1 einen Viewchange-Block platzieren. An den Höhenübergängen von Süden auf Norden mit einer Differenz von 1 werden automatisch Rampen zum Spiel hinzugefügt.

Dynamic Random Levels

Die Levels werden anhand von vorgefertigten Blockmustern zu einem großen Level zur Laufzeit zusammengestückelt, wobei die Reihenfolge zufällig gewählt wird. Erst bei einem kompletten Spielneustart werden die Levels in einer anderen Reihenfolge zusammengefügt.

Highscores

Duelle dich mit deinen Freunden um den Highscore. Mittels Konsolparameter kann ein Spielername übergeben werden (letzter Parameter). Nach jedem Game Over wird in die Highscore-Table geschrieben.

Sphere- & Level-Glow

Je höher die Geschwindigkeit der Spielkugel, desto mehr fängt sie zu leuchten an. Weiters leuchten auch immer jene Blöcke auf den X- bzw. Z-Koordinaten, auf denen sich die Kugel gerade bewegt.

Self implemented Physics

Nun ja, man kann sagen was man möchte. Für eine selbstgeschriebene Physics funktionieren die Kollisionsabfragen verdammt gut. ☺

Startscreen

Der Startscreen bereitet den Spielenden auf das kubische Abenteuer vor ;-).

Female Voices & Game Sound

Während des Spiels ertönen immer wieder Stimmen einer Frau, die entweder

- „Hurry“ (das schwarze Loch wird schneller),
- „Game Over“ (man hat verloren),
- „Again“ (man startet ein neues Spiel,
- „Jackpot“ (man steht in den TOP 10) oder
- „Welcome to BloXphere“ (Startscreen)

sagt. Orientiert haben wir uns hierbei beim Spiel „Super Hexagon“. Apropo: Der Game Sound ist auch von dort, darf aber unter Nennung des Artists verwendet werden (linke untere Spielseite).

Tools

Die Sphere wurde in **Blender** erzeugt und gemapped. Weiters wurde zum Erstellen der Texturen **Photoshop** sowie **Paint** verwendet.

Verwendete Libraries (w/o glew, glfw)

Assimp

Die Assimp Library wurde im Zuge des Objektladens mit in das Projekt aufgenommen. In erster Linie dient es uns zurzeit für das Laden der Sphere (Sphere wurde in Blender erstellt und texturiert). Der Objectloader selbst wurde von http://www.opengl-tutorial.org/beginners-tutorials/tutorial-7-model-loading/ mit leichten Abänderungen übernommen. Die Klasse wurde in Objloader.cpp implementiert.

FreeImage

Diese Library erlaubt uns das Einlesen von Bilddateien (Texturen der Würfel, der Sphere, der Spacemap, …). Verwendet wurde die Library in Texture.cpp.

FMOD

Diese Library liefert uns die notwendigen Funktionalitäten um Sounds im Spiel abzuspielen. Weiters konnte mithilfe dieser Library Beat Detection im Spiel implementiert werden.

Verwendete Tutorials / Papers / Bücher (Referencelist)

Shaderloader

http://www.opengl-tutorial.org/beginners-tutorials/tutorial-2-the-first-triangle/#Shader_Compilation

Objectloader

http://www.opengl-tutorial.org/beginners-tutorials/tutorial-7-model-loading/

Skybox

http://www.mbsoftworks.sk/index.php?page=tutorials&series=1&tutorial=13

Text

https://gitorious.org/wikibooks-opengl/modern-tutorials/source/0b75884b849e8144899b72097d48f7f034c6d4e4:text01_intro/text.cpp
http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-11-2d-text/

Graham Sellers, Richard S. Wright, and Nicholas Haemel. 2013. *OpenGL Superbible: Comprehensive Tutorial and Reference* (6th ed.). Addison-Wesley Professional.

Effects

Mirrors

http://www.songho.ca/opengl/gl_fbo.html
http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-14-render-to-texture/

Graham Sellers, Richard S. Wright, and Nicholas Haemel. 2013. *OpenGL Superbible: Comprehensive Tutorial and Reference* (6th ed.). Addison-Wesley Professional.

Post-Processing-Shader Whirl Effect

http://www.geeks3d.com/20110428/shader-library-swirl-post-processing-filter-in-glsl/
http://adrianboeing.blogspot.co.at/2011/01/twist-effect-in-webgl.html
http://www.mathematische-basteleien.de/spiral.htm

Shadow Maps

http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/
http://ogldev.atspace.co.uk/www/tutorial23/tutorial23.html
http://ogldev.atspace.co.uk/www/tutorial42/tutorial42.html

Graham Sellers, Richard S. Wright, and Nicholas Haemel. 2013. *OpenGL Superbible: Comprehensive Tutorial and Reference* (6th ed.). Addison-Wesley Professional.

Sounds

http://www.roguebasin.com/index.php?title=Implementing_sound_in_C_and_C_Plus_Plus
http://katyscode.wordpress.com/2013/01/16/cutting-your-teeth-on-fmod-part-4-frequency-analysis-graphic-equalizer-beat-detection-and-bpm-estimation/

ParticleSystem

http://ogldev.atspace.co.uk/www/tutorial28/tutorial28.html
https://lva.cg.tuwien.ac.at/cgue/wiki/lib/exe/fetch.php?media=students:cgue14_particle.pdf
http://open.gl/feedback