Antigravity Dokumentation

Jakob Pernsteiner (01627767)
Michael Tischler (01326028)

Beschreibung

Gameplay

Antigravity ist ein Puzzle Platformer, in dem sich der Spieler an Wände teleportieren kann. Ziel ist es über Plattformen ein Portal zu erreichen um jedes der 6 verschiedenen Level erfolgreich abzuschließen.

HUD

User HUD besteht aus alpha geblendeten Text und einfachen Quads für Menüelemente. Die Fonts werden mit der Bibliothek FreeType geladen und in eine Atlas-Texture gerendert. Mit dieser Textur können dann einzelne Glyphs mit einem Offset auf quads gerendert werden. Für Menüelemente wird einfach zuerst ein Quad und dann darauf der Text gerendert.

Frustrum Culling

Für das Frustrum Culling verwenden wird Bounding Boxen, welche beim Import der Meshes berechnet werden. Beim Culling selbst bringen wir die einzelnen Vertices in den Clip-Space damit sich die Koordinaten im Bereich -1 bis 1 befinden. Dann zählen wir ob sich alle Vertices hinter einer der Planes befindet, wodurch das Objekt geculled wird. Sobald sich ein Vertex im Frustrum befindet oder nicht alle Vertices hinter einer Plane sind (Kante könnte das Frustrum schneiden, auch wenn Vertices außerhalb sind) muss das Objekt gezeichnet werden.

Features

PhysX, Player und Gravitations änderungs Mechanismus
PhysX wird für die Physik Simulation verwendet. Der Spieler ist in diese Simulation eingebunden (PhysX Character Controller) und kann sich auf 90° Wände teleportieren, wobei sich auch die Gravitationsrichtung des gesamten Spieles ändert. Dieser Mechanismus wird zum Lösen von Rätsel verwendet, um das Ende jedes Levels zu erreichen. Auch gibt es die Möglichkeit statische und dynamische Physik Objekte zu erzeugen wie zB Würfel und Kugeln, wobei diese auch als kinematische Objekte oder Trigger verwendet werden können. Auch gibt es sogenannte GravSwitches, das sind Schalter die aktiviert werden, wenn die Gravitation in

eine bestimmte Richtung zeigt und deaktiviert, wenn sie in die entgegengesetzte zeigt. Diese können Plattformen de- oder aktivieren.

Beleuchtung

Alle Objekte werden mit einem Directional Light beleuchtet und sind Cel-shaded. Jedes Objekt besitzt ein Material mit Parametern von Diffuse, Specular und Ambient Intensität, sowie Brightness Regelung. Ein Material kann zudem eine Diffuse Textur, welche mittels FreeImage geladen wird, haben oder nur einfarbig sein. Grundsätzlich sind alle Objekte texturiert außer dem Cursor und dem Portal. Auch kann eine Normal-Map einem Objekt hinzugefügt werden, was dann auch für das Cel-Shading verwendet wird. Zudem kann auch eine Emission-Map hinzugefügt werden, welche angibt wo ein Objekt leuchtet. Dies wird dann auch für den Bloom Effekt verwendet. Es können auch mehrere dynamische Point-Lights eingefügt werden. Statische Lichter werden auch in die Lightmap gebaked.

Model Loader

Wir haben einen Model Loader implementiert, der OBJ Modelle (Vertices, Normals, UV Koordinaten) mit der Bibliothek AssImp laden kann und mit diesen Daten ein StaticMesh Objekt erzeugt. Dieses kann dann in unserer Engine verwendet werden.
Zudem wird das gesamte statische Level als fbx Datei importiert, wobei Parameter für die Engine (wie Phyiscs Box Größen, ob man sich hin teleportieren kann, Pfad zu den Lightmaps…) als custom Properties mit importiert.

Hierarchische Animationen

Das Portal weist bereits simple hierarchische Animationen auf, welche aber nur die einzelnen Gameobjects um eine Achse rotiert.

Sounds

Zum Abspielen von Sounds verwenden wir die Bibliothek OpenAL. Sound Dateien werden im wav Format geladen (mit der dr_wav Bibliothek). Eine SoundQuelle wurde als Component realisiert und kann zu jedem GameObject hinzugefügt werden und jederzeit gestartet oder gestoppt werden. Auch können Audioquellen geloopt werden und automatisch beim Initialisieren des Spiels abgespielt werden. Verschiedene Objekte im Spiel haben Sounds, wie das Portal, Sprung-, Teleport- und Todessounds oder auch Menü-Sounds und Hintergrundmusik.

Scene Loader

Die Scene Klasse beinhaltet einen Scene Loader mit dem Szenen aus einer XML Datei geladen werden können. Dabei verwenden wir einfaches Instancing für Shader, Meshes, Texturen, Materialien und Sounds. Diese Können beliebig oft in der Szene verwendet werden und werden aber nur einmal geladen. Zudem werden moving Platforms, Gravitation Switches und Trigger

volumes in der XML definiert. Auch befindet sich in der Szenen Deklaration die Startposition des Spielers und der Ort des Portals, um in die nächste Szene zu kommen.

Scene Manager
Um mehrere Szenen verwalten zu können haben wir einen Scene Manager implementiert, der es uns ermöglicht beliebig zwischen den Szenen, die in XML Dateien gespeichert werden, zu wechseln. Dabei wird bei jedem Wechsel die Szene neue aus der Datei geladen. Eine Szene wird auch automatisch neu geladen, wenn der Spieler out-of-bounds gelangt. Zudem wird die nächste Szene geladen wenn der Spieler in ein Portal tritt.

Menüs
Es gibt in unserem Spiel zwei Menüs, ein Startmenü in dem man das Spiel starten, oder beenden kann und ein Pausemenü in dem man fortsetzen, das Level neu laden oder das Spiel beenden kann. Menüelemente bestehen einfach aus einem Quad und einem Text. Die Farbe des Quads ändert sich wenn man die Maus darüber bewegt und ein Ton wird abgespielt. Beim Click wird auch ein Ton abgespielt.

Effects

Lightmapping mit Texturen

Implementiert von: Jakob Pernsteiner

Zum Baken der Lightmaps verwenden wir das kommerzielle Addon BakeTool[1] für Blender. Dieses erlaubt es uns einfach für viele Objekte automatisch die Lightmaps zu baken und diese in einem Ordner abzulegen. Auch haben wird das Addon etwas modifiziert, damit es uns die Dateinamen als custom Property zu jedem Objekt einträgt, da der Blender Cycles FBX Exporter keine Textur exportiert. Beim FBX Import in unserer Engine laden wir dann mithilfe eines Base Pfades zu dem Lightmap Ordner und der custom Property die Textur uns fügen sie zu jeder StaticMeshComponent hinzu. Beim zeichnen der Objekte wird dann für jeder Objekt die Lightmap gesetzt und im shader dann statt dem normalen Cel-Shading verwendet und einfach zu der diffuse Textur addiert. Die Light maps werden nicht Cel-Shaded, da dies zu zu vielen Artefakten führt. Dynamische point lights müssen aber trotzdem auch berechnet werden.

Normal Mapping

Implementiert von: Jakob Pernsteiner

Zum Normal-Mapping laden wird die Normal Texturen aus unserer XML Szene und fügen sie zu dem jeweiligen Material hinzu. Die Tangents werden von AssImp geim Import der Meshes

[1] BakeTool https://blendermarket.com/products/baketool/

generiert und auch dem Shader übergeben. Die Bitangents werden im Shader berechnet. Normal Mapping generell wird bei uns, so wie unser ganzel Lighting, im View Space berechnet. Die Normals der Texture werden dann einfach normalisiert und mit der TBN Matrix (Tangent, Bitangent und Normal Matrix) multipliziert und für das Lighting verwendet. Normal mapping ist zum Beispiel im ersten Level auf der Kugel und dem Würfel zu sehen.

Quellen:

- https://learnopengl.com/Advanced-Lighting/Normal-Mapping
- http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-13-normal-mapping/

Bloom / Glow Effect

Implementiert von: Jakob Pernsteiner

Für den Bloom Effekt haben wir nicht HDR Rendering verwendet, sondern wir verwenden eigene Emission Texturen für die Objekte, die angeben, wo diese leuchten. Diese werden dann in eine Textur mit der jeweiligen Farbe gerendert und als Eingabe für unseren Bloom-Shader verwendet. Für den Bloom Shader verwenden wir einen eindimensionalen Gauß-Kernel der abwechselnd horizontal und vertikal mit zwei Ping-Pong Framebuffern angewendet wird. Auch verwenden wir das lineares Sampling von OpenGL und weniger Texture fetches zu haben, um somit die Geschwindigkeit zu erhöhen. Nachdem der Gauß-Filter ein paar mal (in unserem Fall 3 mal vertikal und 3 mal horizontal) angewendet wurde. Wird die resultierende Texture zur normalen Color Textur hinzuaddiert, um den gewünschten Effekt zu erzielen.

Quellen:

- https://learnopengl.com/Advanced-Lighting/Bloom
- http://rastergrid.com/blog/2010/09/efficient-gaussian-blur-with-linear-sampling/

Cel-Shading

Implementiert von: Michael Tischler

Cel-Shading wird im Color Fragment Shader realisiert. Dabei quantisieren wir die Faktoren des Diffusen- und Specularteils und zeigen dadurch nur eine bestimmte Anzahl von Farben an. Um schönere Übergänge zu erzielen interpolieren wir den Specularteil durch die smoothstep Funktion zwischen zwei Werte.

Quellen:

- http://prideout.net/blog/?p=22
- http://www.sunandblackcat.com/tipFullView.php?l=eng&topicid=15&topic=Cel-Shading

Edge Detection

Implementiert von: Michael Tischler

Um Outlines für den Toon Effekt zu generieren, verwenden wir in Antigravity Kantendetektion. Der Z-Buffer wird über Framebuffer in eine Texture gespeichert, welche wir dann dem Pixel Shader zur weiterverarbeitung übergeben. Im Shader wird dabei ein Array mit dem aktuellen und dessen Nachbarpixel erstellt, um durch Faltung mittels zwei Sobelkernels die Gradienten der Pixel zu berechnen. Dadurch erhalten wir den Gradienten, speichern den in eine Textur, die mit der Screentexture im Quadshader gemixt wird und erhalten die gewünschten Linien.

Quellen:

- http://www.geeks3d.com/20091216/geexlab-how-to-visualize-the-depth-buffer-in-glsl/
- http://cdn.imgtec.com/sdk-documentation/Edge+Detection.Whitepaper.pdf
- http://prasa.org/proceedings/2007/prasa07-26.pdf

CPU Particles

Implementiert von: Michael Tischler

Das Particle System verwenden wir um die Portale optisch ansprechender zu gestalten. Realisiert wurde es in der Klasse ParticleSystem, welche vom Interface Component erbt und verwendet zwei eigene Shader. Die Klasse Particle speichert dabei alle Informationen der einzelnen Partikel und ein Array speichert dabei alle Partikel. Zwei Schleifen realisieren hierbei den Effekt, in der ersten sehen wir nach, ob ein Partikel im Container nicht genutzt wird und befüllen es mit neuen Informationen und in der zweiten werden Partikel, deren Lebenszeit vorbei ist, “gelöscht”.

Quellen:

- https://learnopengl.com/In-Practice/2D-Game/Particles
- http://www.opengl-tutorial.org/intermediate-tutorials/billboards-particles/particles-instancing/

Tools

Blender

Blender wurde zum Erstellen all unserer Meshes verwendet. Auch unsere statischen Levels wurden komplett in Blender modelliert und im FBX Format exportiert. Alle Objekte besitzen auch UV-Maps und für Lightmapping auch eine zweite UV-Map. Diese wurden teilweise manuell oder

automatisch erzeugt. Lightmaps wurden auch in Blender mit dem Cycles Renderer und dem BakeTool Addon gerendert.

Audacity

Audacity wurde zum schneiden der Sounds und zum konvertieren in das WAV 16-Bit PCM Format verwendet.

Walkthrough

Level 1: “Look Up”

Das Portal befindet sich über dem Spieler, einfach 2-mal an die Wände teleportieren um das Ziel zu erreichen.

Level 2: “Round and Round”

Hier muss sich der Spieler einfach von Plattform zu Plattform teleportieren. Das Portal befindet sich auf einer eigenen Plattform auf der Rückseite der letzten beiden Plattformen.

Level 3: “Huh?”

Hier befindet sich das Portal über dem Spieler, nur wie kommt man dorthin? Es ist möglich sich auf die Seiten der aktuellen Plattform zu teleportieren. Dabei muss sich der Spieler nahe an den Rand stellen bis sich der Teleport-Cursor richtig positioniert und kann sich dadurch auf die Seite der Plattform teleportieren. Danach einfach auf die Plattform mit dem Portal teleportieren oder hineinspringen.

Level 4: “Roll the Ball”

Hier gibt es unsichtbare Wände, wodurch der Spieler eingeschlossen ist. Ziel ist es den Ball durch das Labyrinth mithilfe der Gravitationsänderung zu manövrieren. Dadurch aktiviert sich eine Plattform und schwebt runter Richtung Portal.

Level 5: “Jump and Run”

Dadurch, dass sich der Spieler nicht auf rote Plattformen teleportieren kann, muss er hier seine Jump and Run Skills unter Beweis stellen. Es gibt einen präziseren Sprung, einen fiesen Drop und die letzte kleine Plattform ist nicht erreichbar. Der Spieler darf sich dabei nicht irren lassen und muss sich sofort auf auf die große Plattform teleportieren, wodurch sie aktiviert wird und zu einem neuen Abschnitt fährt. Da muss der Spieler die gleiche Technik wie in Level 3 benutzen

und auf die andere Seite der fahrenden Plattform gelangen, um die richtigen Schalter zu aktivieren. Jetzt bewegen sich die kleinen Plattformen und das der Spieler kann das Portal erreichen.

Level 6: “Up, Up and Away”

Zuerst die Plattform abwarten und die Aussicht genießen, denn es wird ein langer Level. Befindet sich der Spieler sich im Turm, kann er über die kleinen Würfel nach oben zu der ersten Plattform gelangen. Dort kann er über eine weitere Plattform durch springen und teleportieren nach außen gelangen, wo ein weiterer Blöckepart wartet. Ist der Spieler mal am Ende angelangt (es ist wieder ein fieser Drop dabei), befindet sich über ihm ein weiterer Eingang ins Innere des Turms, wo sich eine Plattform um (Sprung mit 180° Drehung plus teleportieren). Dort warten drei kleinere Plattformen und man gelangt durch den Turm auf die andere Seite. Nun fragt man sich, wohin? Unter dem Spieler befindet sich an der Wand eine blaue Plattform, also telepotiert man sich dorthin und gelangt endlich an die Spitze des Turms. Hier kann sich der Spieler bequem von Plattform zu Plattform ins Portal teleportieren.

Level 7: “YOU WIN”

Das Portal ist auf der anderen Seite der Plattform und führt zum Secret-Credit-Level ;)

Controls

Taste	**Aktion**
W,A,S,D	Bewegen
Space	Springen
Linke Maustaste	Teleportieren und Gravitation ändern
F2	FPS und Frametime anzeigen
F3	Wireframe On/Off
F4	Blom On/Off
F5	Normal Mapping On/Off
F8	Frustrum Culling On/Off
F10	VSync On/Off
F11	Fullscreen On/Off

ESC	Menü öffnen/schließen
8	Vorherige Szene laden
9	Nächste Szene laden

Bibliotheken

- ECG Library
- GLEW (http://glew.sourceforge.net/)
- GLFW (http://www.glfw.org/)
- GLM (https://glm.g-truc.net/0.9.8/index.html)
- Nvidia PhysX (https://developer.nvidia.com/physx-sdk)
- FreeImage (http://freeimage.sourceforge.net/)
- FreeType (https://www.freetype.org/)
- AssImp (http://www.assimp.org/)
- OpenAL (https://www.openal.org/)
- dr_wav (https://github.com/mackron/dr_libs)