

CG-UE 2021: Shifter

Submission 2

Konzept, Welt und Spielmechaniken

Der Spieler findet sich auf einer Ebene neben einer Klippe wieder. Ziel ist es, diese Klippe zu erklettern. Stürzt der Spieler in den Abgrund, hat er verloren und wird zum letzten Wiedereinstiegspunkt zurückgesetzt. Wiedereinstiegspunkte werden dabei während des Fortschreitens im Spiel immer wieder durch das Erreichen von Checkpoints gesetzt.

Grundsätzlich kann der Spieler dabei gehen, laufen und springen. Auch während des Fallens ist es eingeschränkt möglich, die Richtung zu ändern, wobei Richtungsänderungen mit dem Tasten (z.B. `strafe_right` halten) nur langsam die Spielergeschwindigkeit ändern, Blickrichtungsänderungen die aktuelle Geschwindigkeit nicht beeinflussen. Damit ist es grundsätzlich möglich, die meisten Hindernisse auf langsame, dafür sichere Art zu überqueren, oder aber unter vermehrter Verwendung von Blickrichtungsänderungen schneller das Level zu schaffen.

Zusätzlich zu diesen grundlegenden Bewegungen gibt es noch den Dimensionswechsel. Durch Halten der linken oder rechten Maustaste kann der Spieler in zwei andere Dimensionen wechseln. Diese Dimensionen unterscheiden sich einerseits visuell voneinander aber auch physikalisch: Manche Plattformen und dynamische Objekte sind nur in bestimmten Dimensionen vorhanden, weshalb der Spieler oft gezwungen ist (zumindest kurzfristig) in eine andere Dimension zu wechseln, um weiter vorankommen zu können. Kollisionen sind ausschließlich innerhalb von Dimensionen möglich.

Aktueller Stand / Noch bis zum Spiele-Event geplant

Verwendete Bibliotheken

- Physik-Engine: NVIDIA PhysX 4.1
<https://developer.nvidia.com/physx-sdk>
- FreeType 2 für Font-Loading und Glyph-Rendering (in Bitmaps)
<https://www.freetype.org/freetype2/docs/index.html>
- Konfigurationsdatei lesen: `inih`
<https://github.com/benhoyt/inih>
- Modelle und Szene laden: `tinyglTF` (header-only)
<https://github.com/syoyo/tinyglTF>

CG-relevante Features

Gameplay

HUD

Mithilfe von FreeType können Fonts in das Framework geladen werden. Über unsere Rendering-Schnittstelle können ASCII-Zeichenketten dann angezeigt werden. Außerdem ist es möglich, angezeigten Text langsam ausblenden zu lassen.

Collision Detection and Advanced Physics

Als Physik-Engine wird PhysX verwendet. Spielerbewegung ist mithilfe der Character-Controller-API von PhysX umgesetzt. Geladene Spiel-Objekte, wie Terrain beispielsweise, werden je nach Einstellung direkt in der Physik-Engine widerspiegelt, damit der Spieler sich nicht durch diese hindurchbewegen kann.

Dynamische Objekte werden für bestimmte Event-Sequenzen verwendet. Eine Event-Sequenz am Anfang des Spiels ist, dass ein Felsen von der Klippe herabstürzt und eine Plattform zerstört.

Wir verwenden außerdem Trigger für verschiedene Ereignisse:

- Beim Sturz in den Abgrund wird der Spieler auf den Wiedereinstiegspunkt zurückgesetzt.
- Beim Erreichen eines Checkpoints wird der Wiedereinstiegspunkt an den Ort des Checkpoints gesetzt und ein Text wird angezeigt.
- Beim Erreichen des Ziels wird ein Text angezeigt.
- Beliebige Events: Vor dem Erreichen der zweiten Holzplattform befindet sich ein Trigger, welche den Sturz eines Felsens auslöst, der die Plattform zerstört – damit muss der Spieler zum ersten Mal die Dimension wechseln, um weiterzukommen.

View Frustum Culling

Da unsere Szene viele Objekte umfasst, berechnen wir das View-Frustum der Kamera, um mithilfe des KD-Baums, jene Objekte von render-Aufrufen auszuschließen, die zurzeit nicht sichtbar sind. Die Berechnung des Cullings erfolgt über 6 Ebenen, welche aus der View-Projection Matrix generiert werden. Nur wenn sich ein Teil der Bounding Box innerhalb von allen 6 Ebenen befindet, wird das Objekt gerendert.

Zurzeit funktioniert dieses Feature noch nicht komplett; bei bestimmten Blickrichtungen stimmen die ignorierten Objekte nicht mit denen der tatsächlichen Blickrichtung überein.

Nachdem wir Instancing als Rendering-Technik eingebaut haben, hatte das Aktivieren von View-Frustum Culling sogar eine negative Auswirkung auf die Performance.

Effekte

Bloom

Wir haben eine simple, aber flexible Post-Processing-Pipeline als Teil der Engine entwickelt, wobei Zwischenergebnisse jeweils in einzelne Framebuffer gerendert und das finale Ergebnis schlussendlich als Textur auf einem, den gesamten Viewport überspannenden Rechteck gerendert wird (das Aktivieren des Wireframe-Modus zeigt daher nicht mehr die eigentliche Geometrie der Szene sondern nur noch ebenjenes Rechteck an) .

Innerhalb dieser Pipeline haben wir einen Bloom-Effekt umgesetzt: im ersten Schritt werden helle Pixel herausgefiltert und mit Gaussian Blur weichgezeichnet, wobei wir Gaussian Blur aus Performancegründen hier als *2-pass iterative gaussian blur* umgesetzt haben ($O(2n)$ vs. $O(n^2)$). Die entstehende Textur blenden wir anschließend additiv über die Originalszene.

Vertex Shader Animation

Der Shader, welcher für Physically Based Rendering zuständig ist, kann zusätzlich Vertices aufgrund einer trigonometrischen Funktion, der ursprünglichen Position sowie der verstrichenen Zeit vertikal versetzen. Dies wird verwendet, um Oberflächen wie Wasserwellen zu animieren. Die Oberflächennormale sind dabei jene Vektoren, die normal auf die Gradienten (in x/z Richtung) der Wellenfunktion stehen. Die Wellenfunktion kann mittels Uniforms parameterisiert werden.

Physically-Based Rendering

Für die Lichtberechnung verwenden wir ein physikalisch Basiertes Berechnungsmodell. Das Format der Map, gltf unterstützt nur den Export von der Roughness und Metallic Parameter daher verwenden wir ebenfalls dieses Modell. Unsere implementation basiert auf dem GLTF Referenz Shader, die Berechnung des Diffusen Lichts wurde vom Disney BRDF Modell übernommen da es einen besseren Bildeindruck ergibt.

Der größte optische Unterschied zum regulären Lichtmodell liegt in den Specular Reflections, insbesondere sichtbar anhand Fresnel Reflektionen. Diese lassen sich für bestimmte Materialien, wie beispielsweise Wiesen oder Pflanzen auch deaktivieren.

KD-Tree

Der KD-Tree teilt an jedem Knoten den Objectspace in zwei Teile auf. Die Teilungen kann an einer beliebigen Stelle entlang einer der drei Koordinatenachsen stattfinden. Um einen möglichst balancierten Baum zu bekommen, testen wir beim Erstellen des Baumes immer alle möglichen Positionen zwischen Objekten an allen Achsen. Am Ende wird jene Ebene gewählt, bei welcher die Objekte möglichst gleichmäßig auf die Kinder des Knotens verteilt werden.

Tastenbelegung

| Character Steuerung | |
|---------------------------------|------------------|
| Vorwärts bewegen | W |
| Rückwärts bewegen | S |
| Nach links bewegen | A |
| Nach rechts bewegen | D |
| Springen | Space |
| Laufen | Shift Links |
| Kamera Bewegen | Maus |
| Wechsel in Dimension 2 | Linke Maustaste |
| Wechsel in Dimension 3 | Rechte Maustaste |
| Noclip Steuerung | |
| Vorwärts bewegen | W |
| Rückwärts bewegen | S |
| Nach links bewegen | A |
| Nach rechts bewegen | D |
| Nach oben fliegen | Space |
| Nach unten fliegen | Strg Links |
| Schneller fliegen | Shift Links |
| Sonstige Steuerung | |
| Beenden | ESC |
| <i>Wireframe einschalten*</i> | <i>F1</i> |
| Backface Culling umschalten | F2 |
| Noclip umschalten | F3 |
| View Frustum Culling umschalten | F8 |
| Helligkeit erhöhen | Numpad + |
| Helligkeit senken | Numpad - |

**aufgrund der Post-Processing Effekte funktioniert die Wireframe Darstellung nicht mehr und es wird ein weißes Bild angezeigt.*

Level

Die Karte ist aktuell noch unfertig, allerdings werden in einem kurzen Level sämtliche Effekte und Spielmechaniken implementiert.

Verwendete Materialien

- Font-Loading und Text-Rending:
<https://learnopengl.com/In-Practice/Text-Rendering>
<https://www.freetype.org/freetype2/docs/tutorial/index.html>
- Bloom-Effekt:
<https://learnopengl.com/Advanced-OpenGL/Framebuffers>
<https://learnopengl.com/Advanced-Lighting/Bloom>
- Physics:
<https://gameworksdocs.nvidia.com/PhysX/4.1/documentation/physxguide/Manual/Index.html>
- Vertex Shader Animation:
<https://catlikecoding.com/unity/tutorials/flow/waves/>
- Physically-Based Rendering:
<https://github.com/KhronosGroup/glTF-Sample-Viewer/tree/master/source/Renderer/shaders>

Burley, B., & Studios, W. D. A. (2012, August). Physically-based shading at disney. In *ACM SIGGRAPH* (Vol. 2012, pp. 1-7). vol. 2012.
- View Frustum Culling / KD-Tree:
<http://www.lighthouse3d.com/tutorials/view-frustum-culling/geometric-approach-extracting-the-planes/>