GT2

ein prozedurales Rennspiel in vvvv

von Stefan Kernjak

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Aufgrund der Lesbarkeit wurde auf das allgemein feminine: /Innen verzichtet und nur die maskuline Bezeichnung verwendet.

Betreuer der Bachelorarbeit: DI (FH) Thomas Radeke

Abstract

DE

GT2 ist die Weiterentwicklung eines Rennspieles in vvvv, das ursprünglich im Fach “Projektarbeit Interaction Design” gemacht wurde. Ziel des Spiels ist es, einen hohen Highscore zu erreichen, indem man möglichst lange Hindernissen auf einer prozedural generierten Strecke ausweicht.

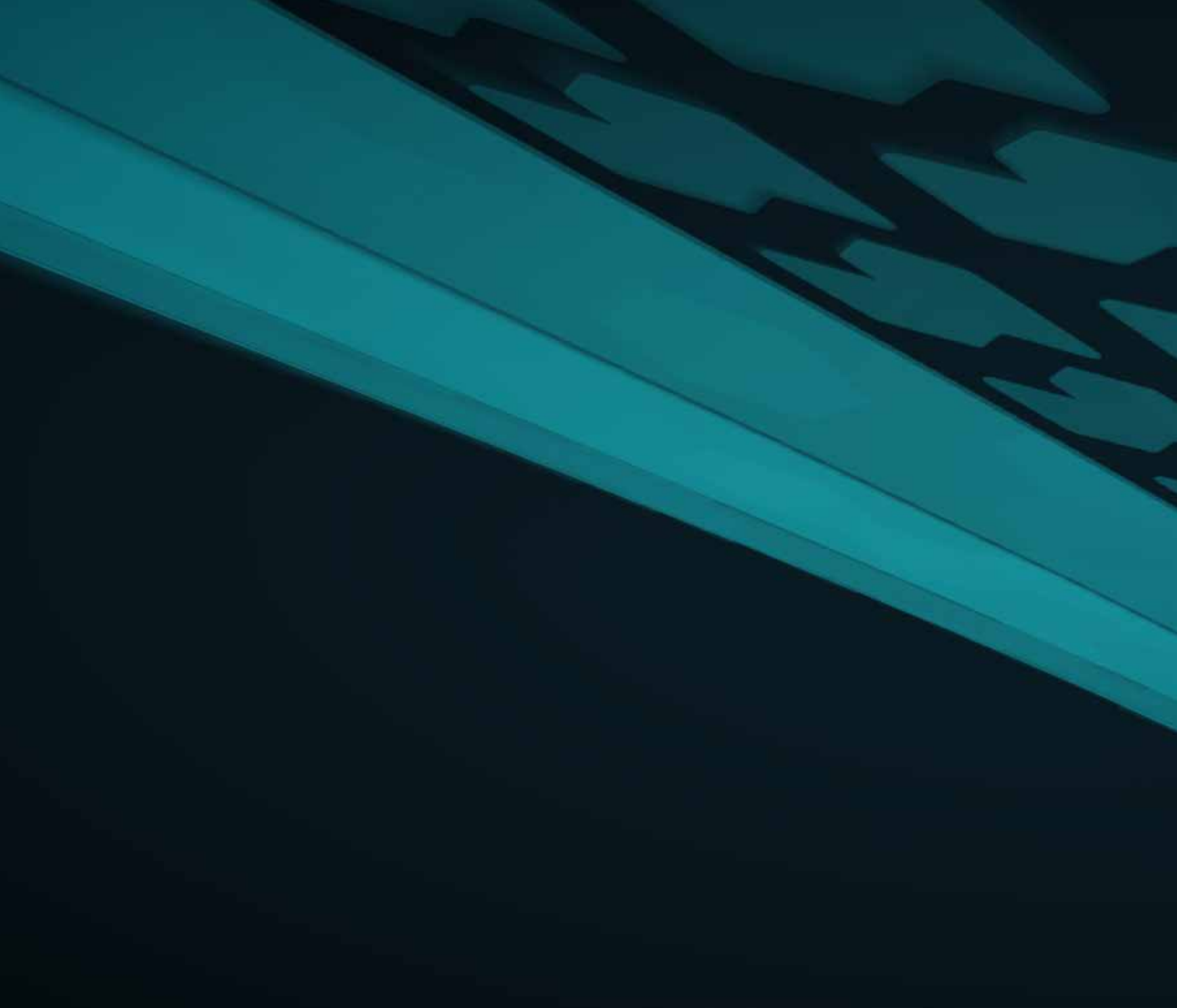
In der vorliegenden Arbeit wird beschrieben, was sich warum verändert hat und wie die Veränderungen umgesetzt wurden.

EN

GT2 is the progression of a racing game in vvvv, based on a project done in “Projektarbeit Interaction Design”. The goal of the game is to reach as high a highscore as possible, by avoiding barricades along a procedurally generated track.

This work describes how and why changes were made.

Inhaltsverzeichnis



Einleitung

Ursprüngliche Idee und Umsetzung

Die erste Version entstand im Fach "Projektarbeit Interaction Design" bei Daniel Fabry in Zusammenarbeit mit Stefan Kaltenegger, Zrinko Kozlica und Tobias Zotter. Unsere Idee war, mit Hilfe einer großen 3D Projektionsfläche und Microsoft Kinects mehrere Spieler in einem futuristisch aussehenden Rennspiel selbst zum Fahrzeug werden zu lassen, indem sie durch Körpereinsatz den Hindernissen ausweichen müssen.
Da die Steuerung über mehrere Microsoft Kinects nicht präzise genug war, wurde dann doch nur eine Microsoft Kinect verwendet und ein Fahrzeug auf die Rennstrecke gesetzt (siehe Abb. 1).

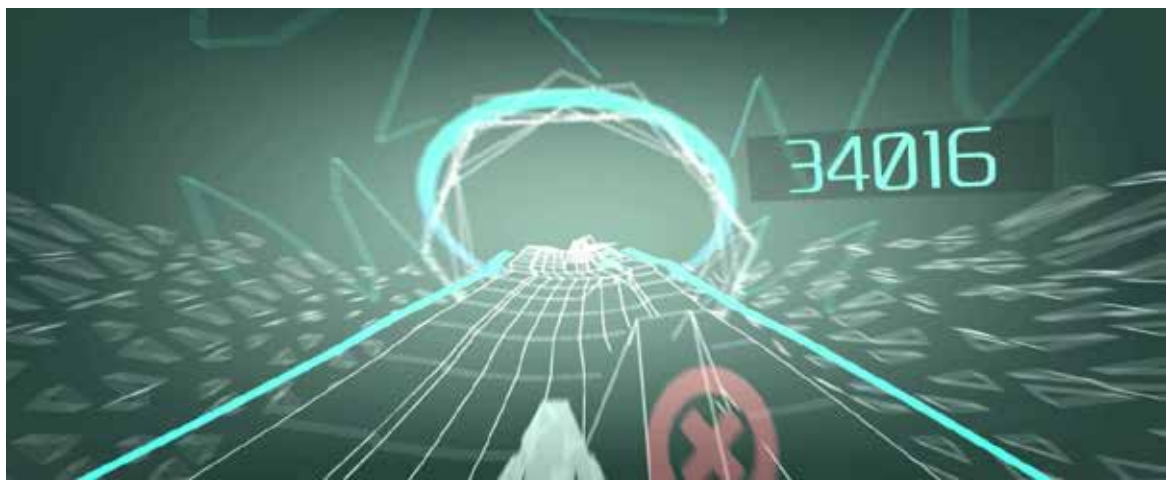


Abb. 1: Erste Version

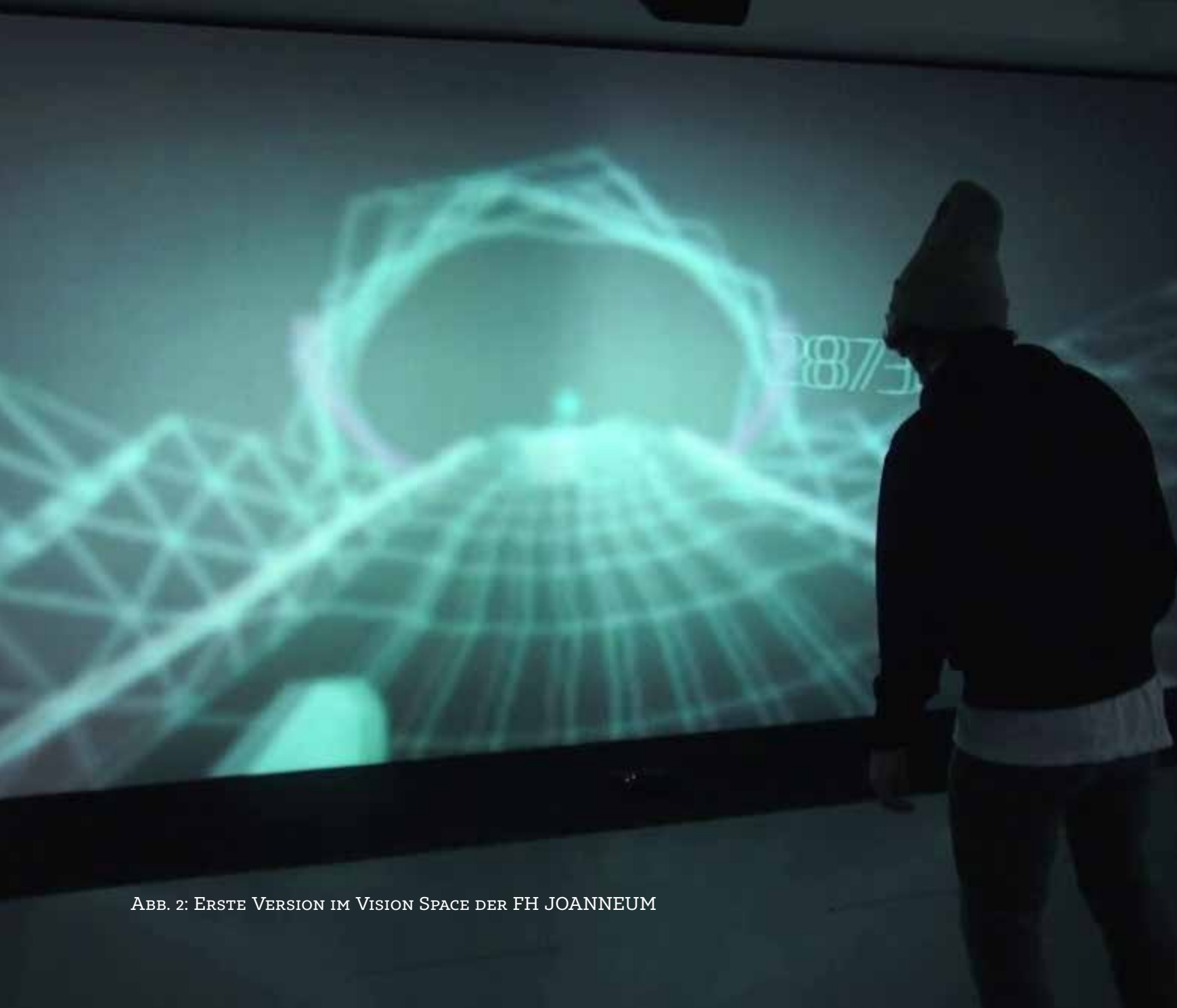


Abb. 2: Erste Version im Vision Space der FH JOANNEUM

vvvv – a multipurpose toolkit

vvvv beschreibt sich selbst als „a graphical programming authoring environment".

An Stelle von Operatoren und Funktionen gibt es Knoten, so genannte Nodes, die visuell mit Linien verbunden werden (siehe Abb. 3), ähnlich wie Max oder Pure Data. Im Gegensatz zu klassischen prozeduralen oder funktionalen Programmiersprachen können Veränderungen am Code während der Laufzeit durchgeführt werden und sind damit sofort sichtbar (vgl. vvvv group 28.8.2013, online), was sehr hilfreich bei Interaktions- und visuellem Design ist.

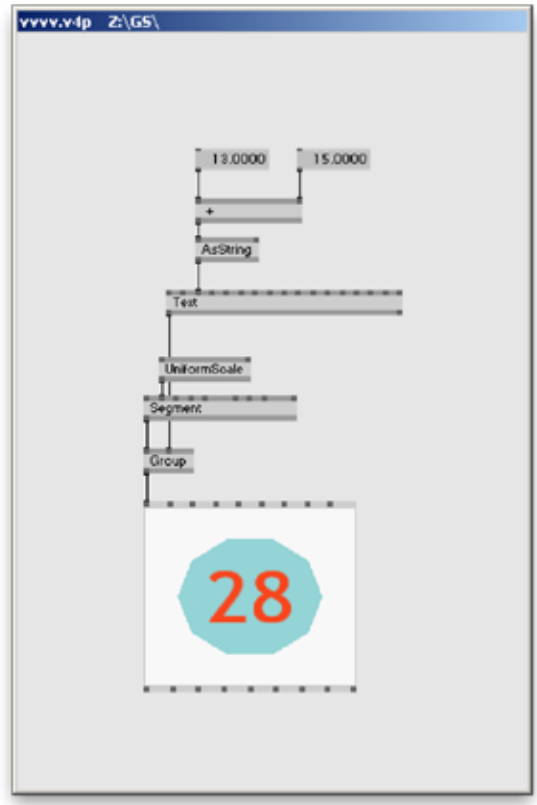


Abb. 3: vvvv Interface

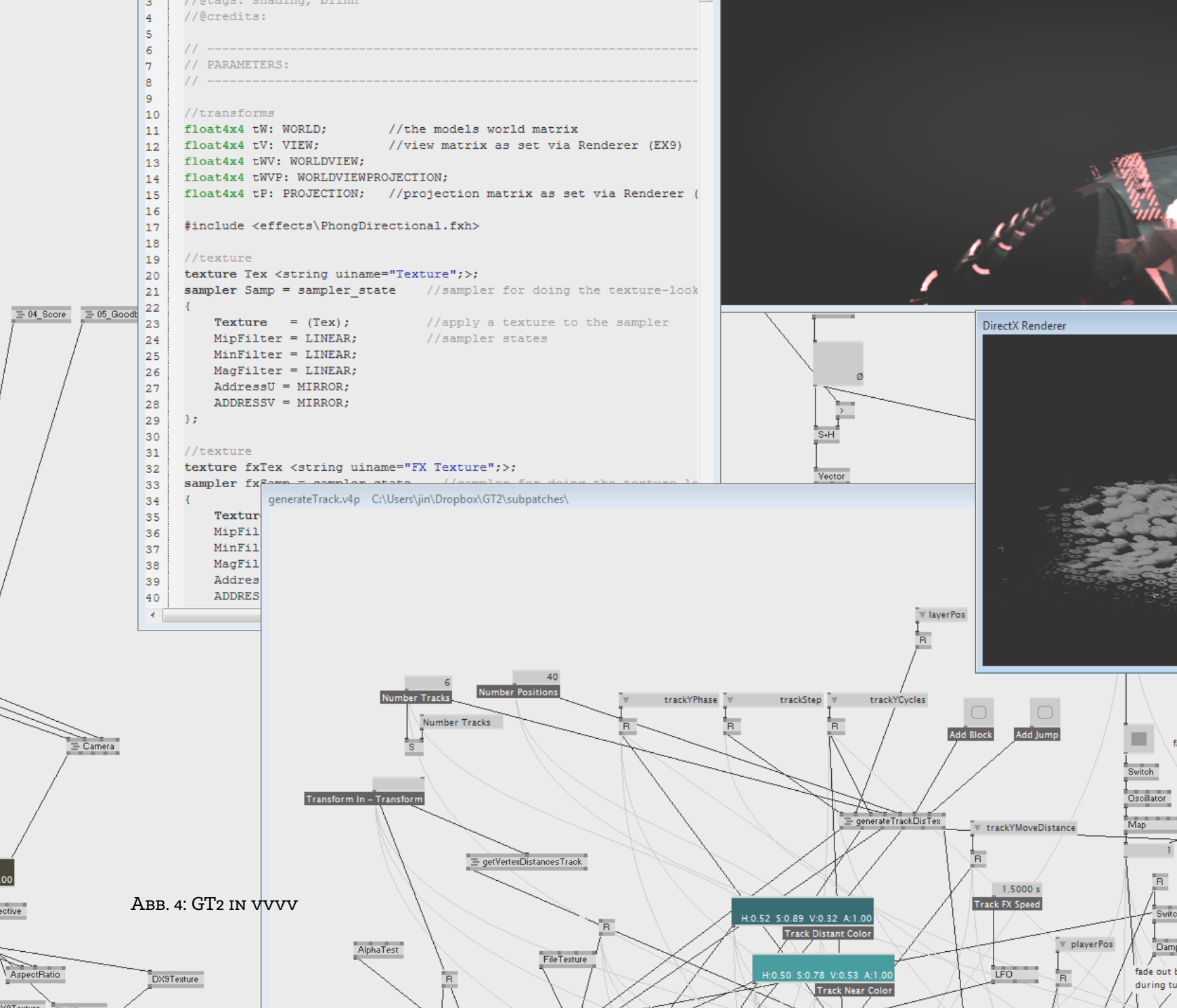
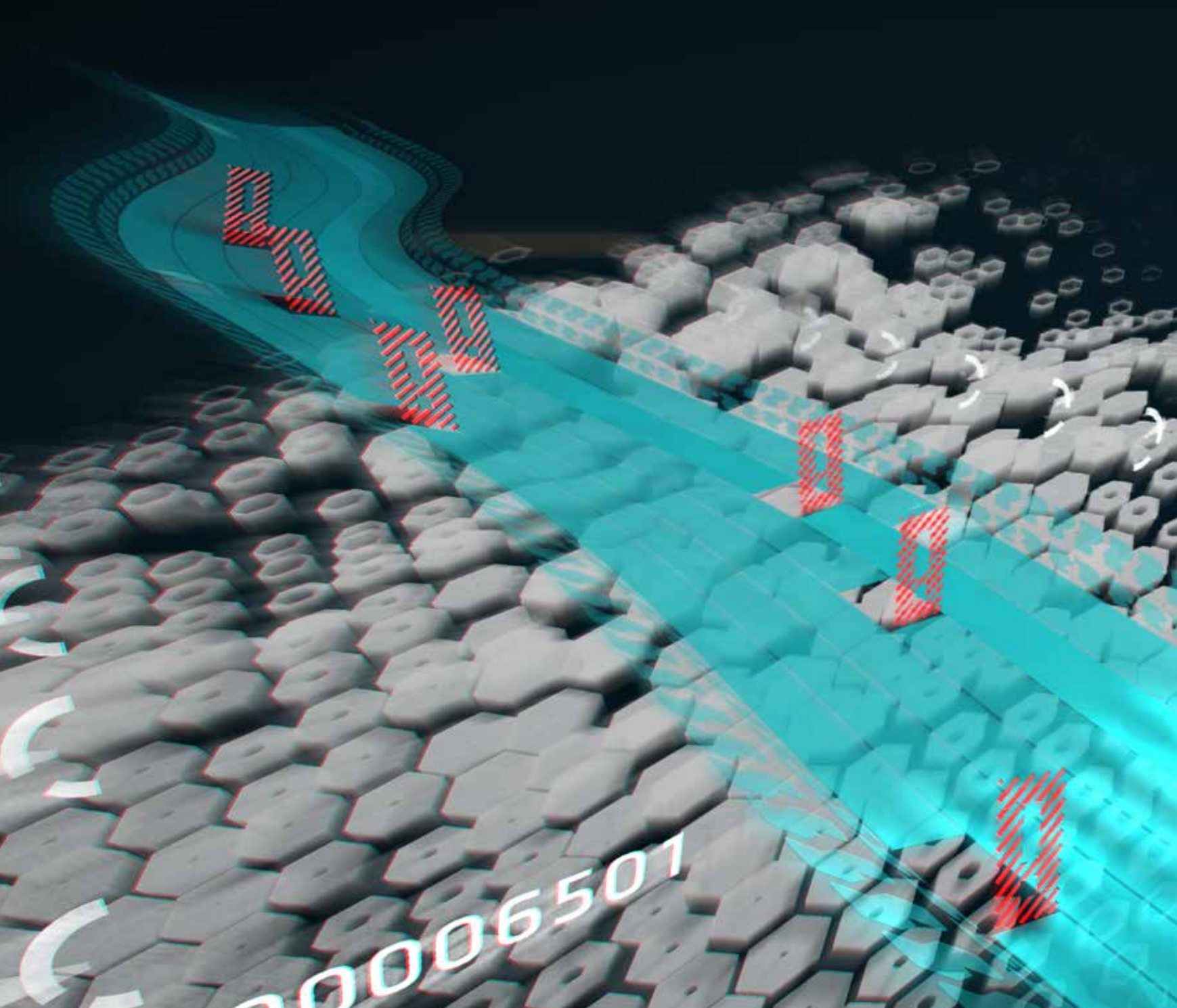


Abb. 4: GT2 in vvvv


00006501

Veränderungen

Basierend auf den Benutzerreaktionen bei der Präsentation des Projekts sollte alles benutzerfreundlicher und visuell interessanter gestaltet werden.

Spieler hatten meistens Probleme mit den knappen Anweisungen und der Steuerung. Nach dem Spiel kam dann der frustrierende Game Over Screen. Die meisten warteten den Highscore Screen danach nicht mehr ab. Das Unterschreiben vom Highscore verstand kaum jemand und die Unterschriften identifizierten die Spieler schlussendlich nicht eindeutig.

An den Anfang des Spieles wurde nun eine ausführliche Einführung hinzugefügt. Als erstes wird dem Spieler geholfen in eine gute Position vor dem Microsoft Kinect zu kommen. Dann werden schrittweise die Steuerung und die Spielmechaniken erklärt.

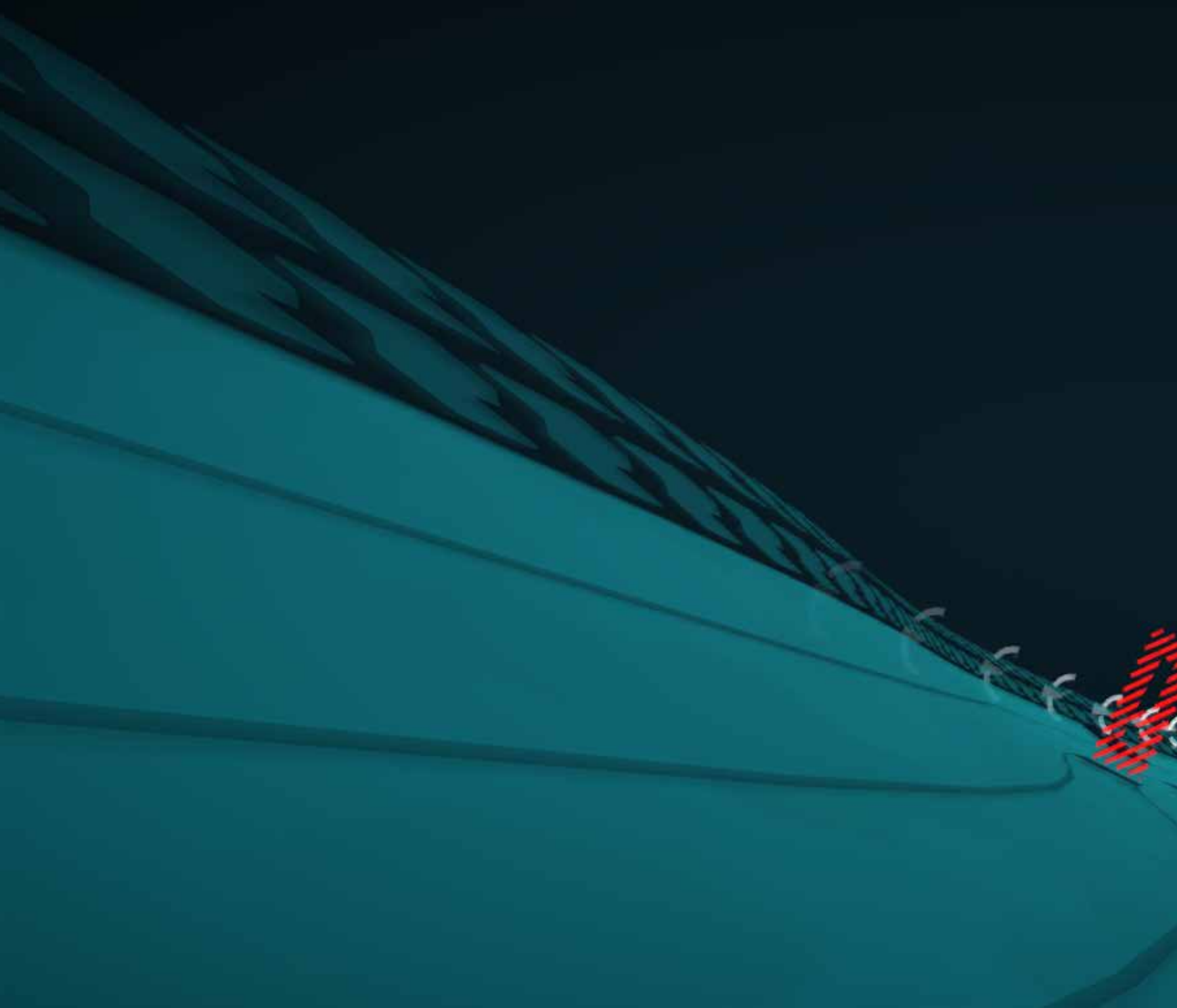
Die Strecke war ursprünglich darauf ausgelegt, dass sich ein Spieler virtuell auf ihr befindet und damit an die Unterkante des Bildschirms gebunden. Dieser Aspekt fällt nun komplett weg. Strecke und Kamera sind viel freier in ihren Bewegungen und Gestaltung, was den Spieler stärker fordert und die Strecke nicht mehr zwingt komplett gerade vor dem Spieler zu sein.

Um ein höheres Geschwindigkeitsgefühl zu vermitteln bewegt sich nun auch die Landschaft mit.

Der Ablauf wurde außerdem dahingehend verändert, dass das „Game Over” viel weniger betont wird und das Unterschreiben

des eigenen Highscores komplett entfernt wurde. Dadurch wird der Spielablauf flüssiger, freundlicher und schneller.

Außerdem gibt es jetzt auch noch Bonuspunkte, wenn man knapp an Hindernissen vorbeifährt oder ihnen über eine längere Zeitspanne ausweicht, was dem ganzen Spiel mehr Tiefe gibt.



Spielablauf

Warten auf Spieler

Solange sich kein Spieler im vordefinierten Bereich befindet, zieht die Landschaft vorbei und im Vordergrund schwebt „Come Closer". Sobald ein Spieler für kurze Zeit nahe genug ist, startet das Spiel (siehe Abb. 5).

Abb. 5: Warten auf Spieler

Spieleinführung

Alle Einführungsschritte werden durch einen kurzen, im Raum schwebenden Text erklärt. Am Anfang ist der Text weiß, mit dem Erfüllen der Einführungsaufgabe wird dieser langsam gelb und springt als Bestätigung nach vorne. Sollte der Spieler etwas falsch machen, bewegt sich der Text, angelehnt an die Bewegung beim Kopfschütteln, schnell seitlich hin und her.

Ein gelber Zylinder vertritt während der Einführung als abstrakter Avatar den Spieler in der Welt. Da dieser auf die Bewegungen des Spielers reagiert, sollte die Beziehung zwischen ihm und dem Avatar schnell nachvollziehbar sein. An Stellen, wo der Spieler seinen Avatar an einen konkreten Punkt bewegen soll, werden diese Bewegungen durch gelbe, animierte Pfeile vorgegeben.

Es folgt eine Erläuterung der einzelnen Kapitel der Spieleeinführung.

Position

Nachdem ein Spieler erkannt wurde, bewegt sich die Kamera zur Positionierungshilfe. Hier wird der Spieler zu einer guten und mittigen Postion vor dem Microsoft Kinect angeleitet (siehe Abb. 6).

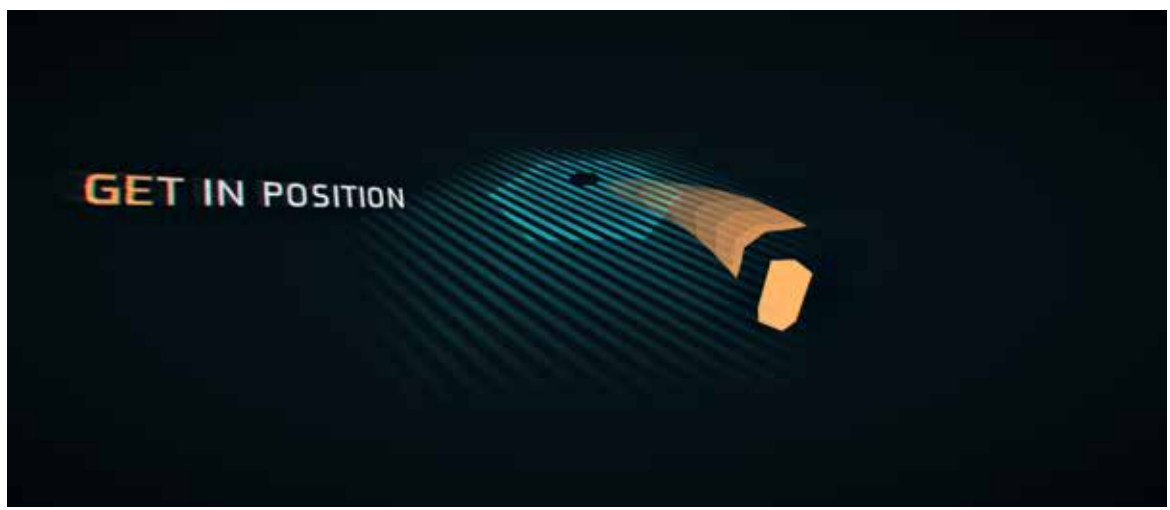


Abb. 6: Position

Bahnenwechsel

Hier wird dem Spieler die grundsätzliche Steuerung erklärt. Gelbe Richtungspfeile deuten wie zuvor die Richtungen an, in die sich der Spieler bewegen sollte. Wobei sich die Pfeile nur bei seitlichen Bewegungen verändern, Vorwärts- und Rückwärtsbewegungen haben keinen Effekt.
Je nach Position des Spielers verschiebt sich die hell beleuchtete Bahn entsprechend der Position des Avatars. Nach 15 Bahnenwechseln geht es weiter (siehe Abb. 7).

Abb. 7: Bahnenwechsel

Sprünge

Da die Steuerung zum Wechsel zwischen den einzelnen Bahnen nun bekannt sein sollte, gibt es hier keine vorgegebenen Richtungspfeile mehr. Die Kameraposition verändert sich um zu vermitteln, dass diese keinen direkten Einfluss auf die Steuerung hat. Dem Spieler soll vermittelt werden, dass er über die auf der Bahn verteilten Sprungrampen fahren sollte, um später auch durch Sprünge den Hindernissen ausweichen zu können. Um das Ganze unterhaltsamer zu machen schlägt der Avatar fliegend Saltos (siehe Abb. 8).

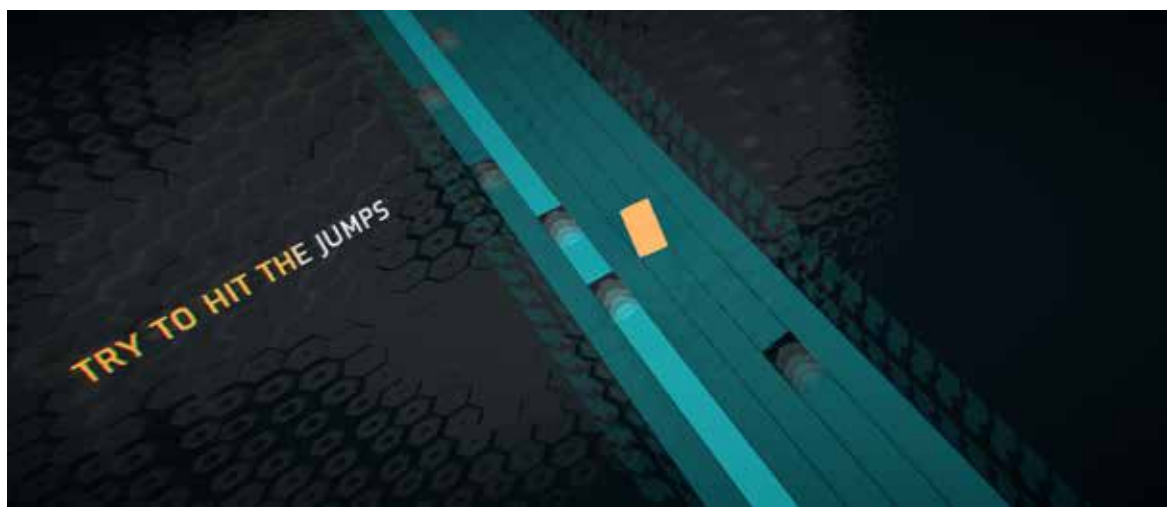


Abb. 8: Sprünge

Hindernisse

Die primäre Spielmechanik, dass man Hindernissen ausweichen muss, wird hier erklärt. Gleich wie bei den Sprüngen gibt es keine Pfeile und die Kameraposition verändert sich wieder (siehe Abb. 9).

Abb. 9: Hindernisse

Spiel

Der Übergang von der Spieleinführung zum eigentlichen Spiel sollte einen gewissen Wow-Effekt haben. Die Kamera befindet sich direkt über der Strecke und hinter dem Avatar des Spielers, die Welt verändert ihr Aussehen und eine treibende Musik setzt ein. Alles, was ein Spieler in der Spieleeinleitung gelernt und geübt hat, wird nun angewendet (siehe Abb. 10).

Mit der Zeit wird die Fahrt immer schneller und das Ausweichen der Hindernisse zunehmend schwieriger, bis man es nicht mehr schafft den Hindernissen auszuwei-

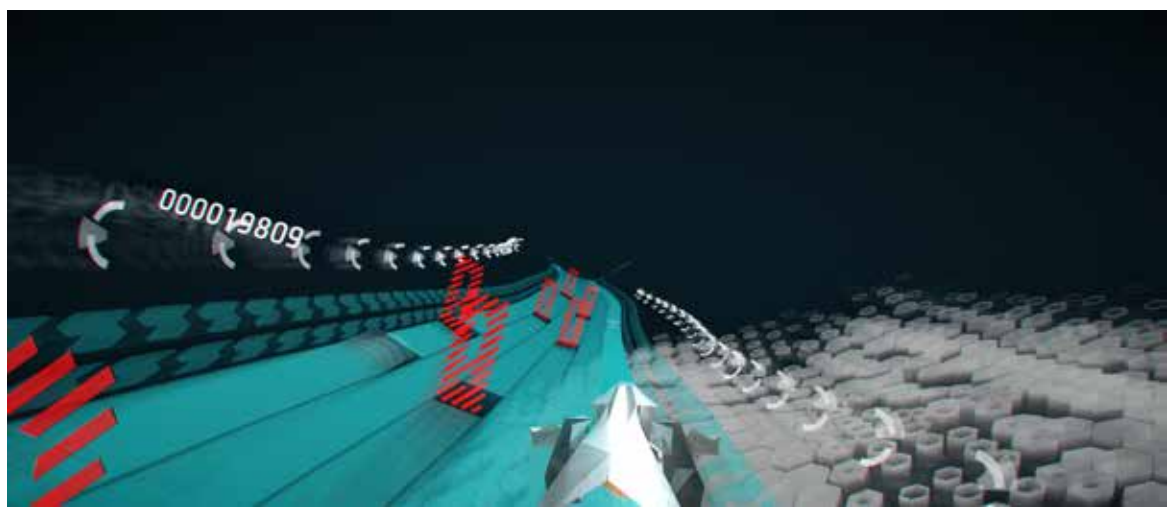


Abb. 10: Während dem Spiel

chen. Ähnlich alten Arcade-Games hat das Spiel damit kein richtiges Ende und das Ziel ist es selbst besser zu werden und vorherige Highscores zu überbieten.

Niedrigere Gesundheit wird durch eine etwas höhere Fahrgeschwindigkeit, ein kontrastreicheres, rötlicheres Bild, eine dunkle Vignette und hörbaren Puls vermittelt.

Das Spiel endet, wenn man zu viel Schaden genommen hat (siehe Abb. 11).

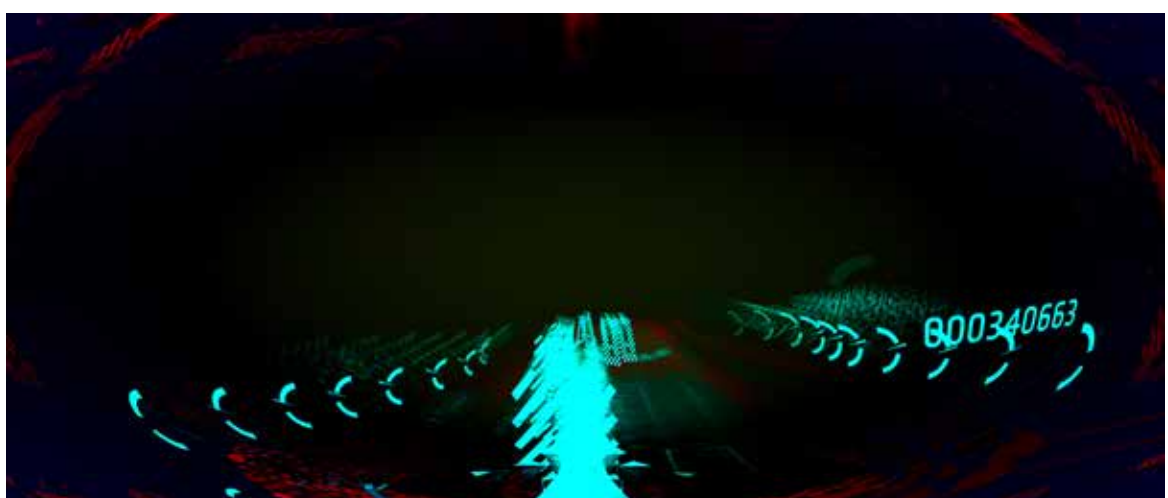


Abb. 11: Das Fahrzeug fuhr in ein Hindernis

Highscore

Die Kamera macht wieder einen Flug über die Landschaft, bis man bei dem eigenen Highscore angelangt ist, der langsam die erreichten Punkte hinaufzählt. Aus der Ferne fliegen die derzeitigen Highscores auf einen zu. Sobald man einen Highscore überbietet fliegt dieser weiter. So arbeitet man sich langsam die Highscore-Liste hinauf (siehe Abb. 12).

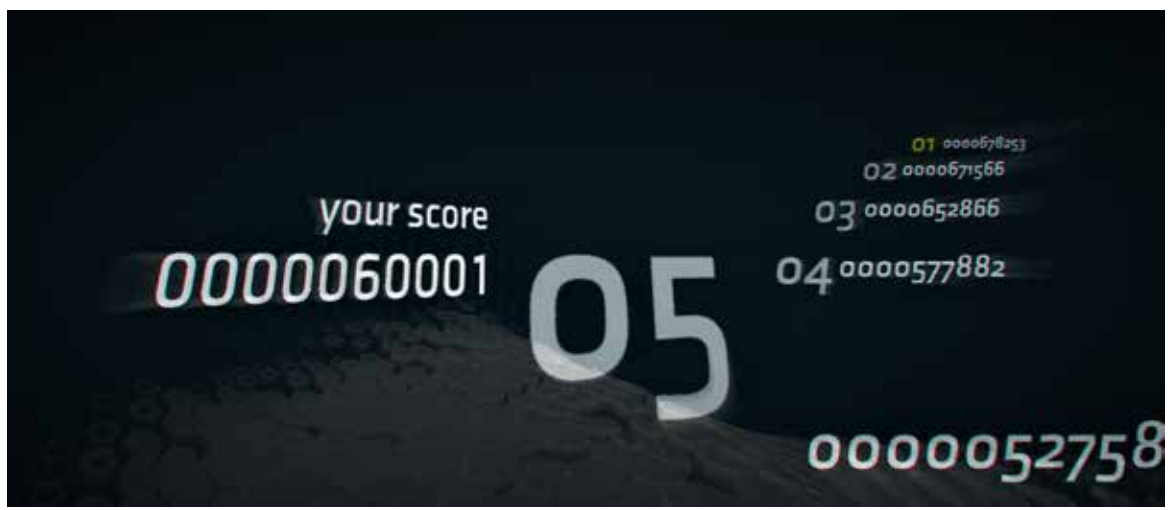
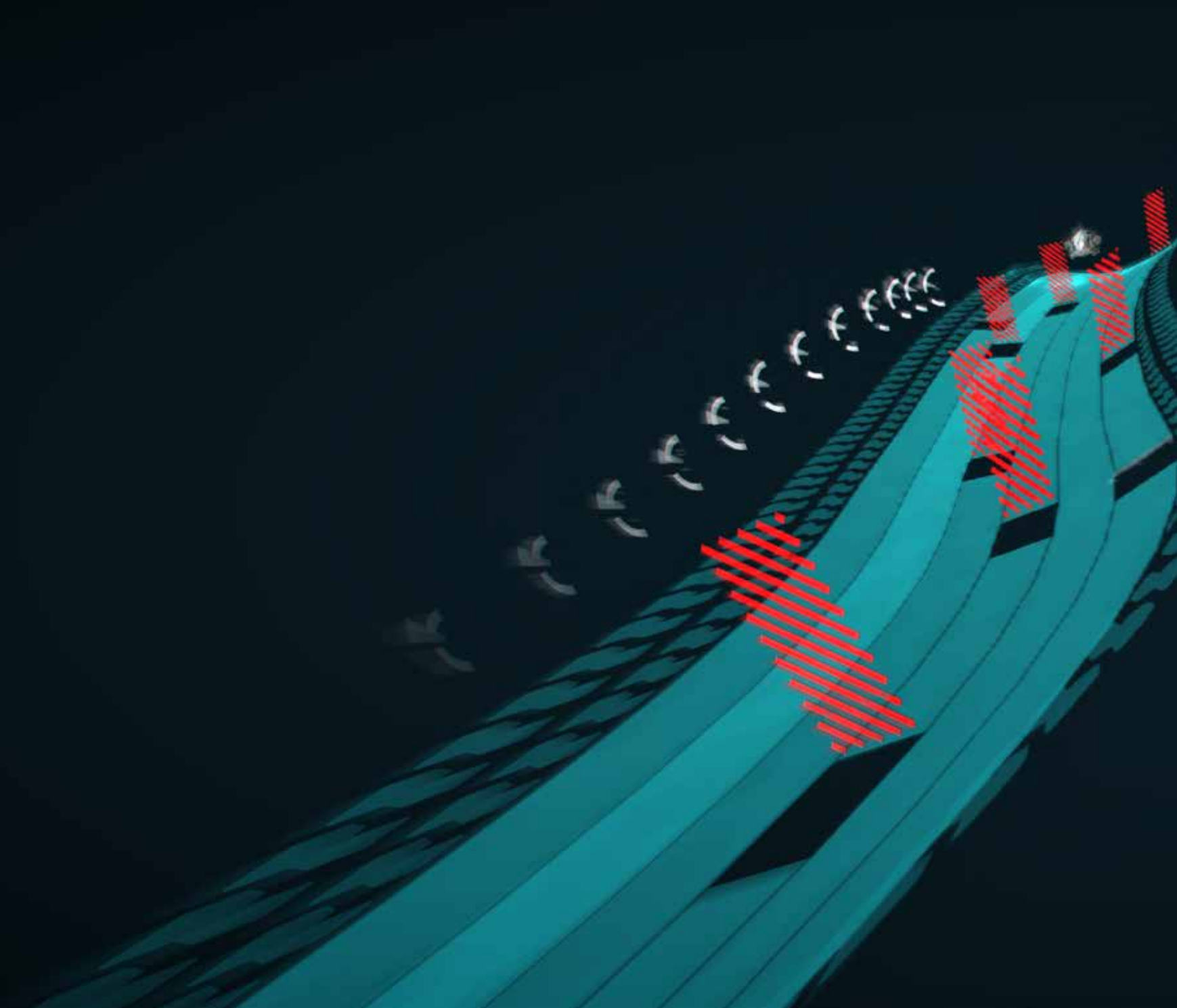


Abb. 12: Highscore

Verabschieden

Nach dem Highscore verabschiedet sich das Spiel mit einem erneuten Kameraflug und dem Text „Thanks for Playing". Danach wartet das Spiel wieder auf einen neuen Spieler (siehe Abb. 13).

Abb. 13: Thanks for Playing



Gestaltung

Grundsätzlich, im Vergleich zur ersten Version, sollte das Spiel ein zugänglicheres und klareres Design und eine retro–futuristische Anmutung haben und an Computersimulationen erinnern.

Den Spielern werden in der neuen Version die Mechaniken erklärt. Das „Game Over" hat viel weniger Bedeutung. Wo früher Wireframes waren, sind nun gefüllte Flächen. Strecke und Landschaft sind damit schneller und leichter erkennbar. Die Schrift ist auch weniger streng und techy.

Die UI Elemente sollten interessanter sein, da mehr Wert auf die Verteilung in unterschiedliche Tiefenebenen gelegt wurde und diese nun räumlich Teil der Spielewelt sind.

Die Gestaltung der einzelnen Elemente geschah zum großen Teil direkt in vvvv. Das grundsätzliche Design stand bereits und da vvvv sehr rapid-prototyping-freundlich ist, konnten schnell unterschiedliche Variationen ausprobiert werden.

Das finale Aussehen ergab sich aus dem Zusammenspiel zwischen Design, technischen Fähigkeiten und Performance.

Der Fahrzeug/Bike Mesh wurde mit Augenmerk auf eine interessantere Silhouette hin überarbeitet. Außerdem gibt es auch Animationen für Bahnenwechsel, Crashes und Sprünge, was den Spielern mehr Feedback gibt und sich damit besser anfühlt.

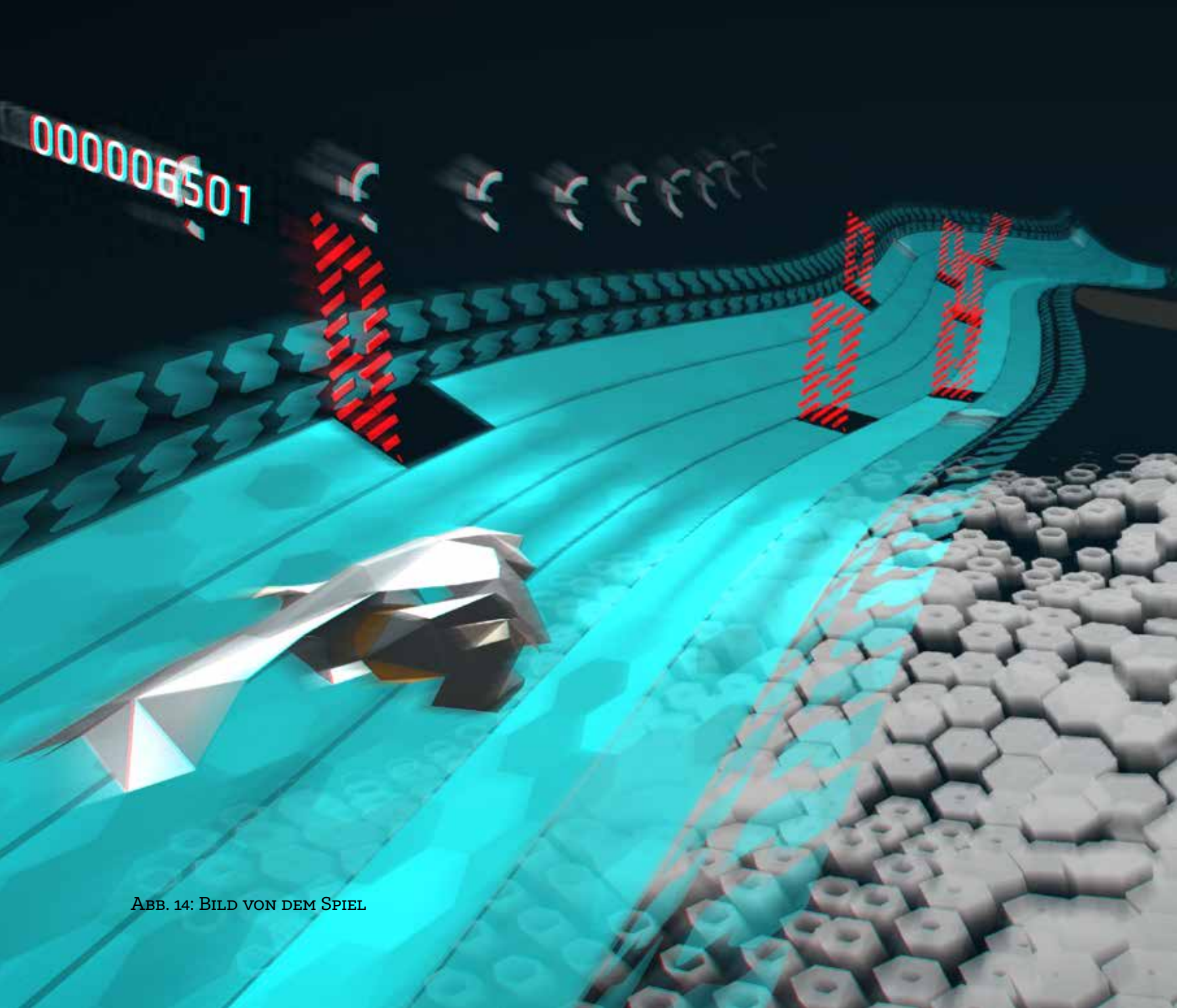
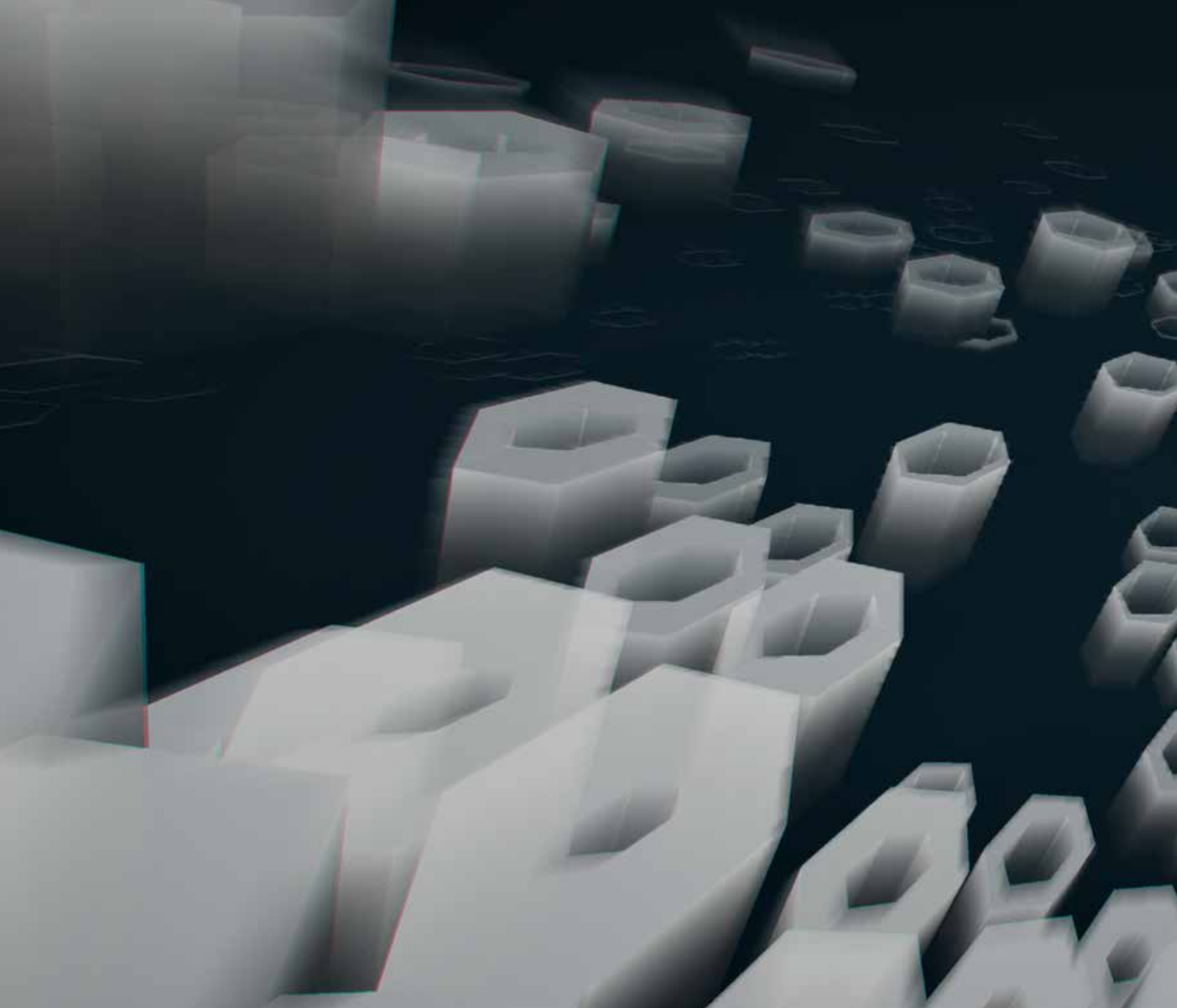


Abb. 14: Bild von dem Spiel



Umsetzung

Begriffserklärungen

Shader

Code, der beschreibt, wie etwas dargestellt werden soll. Er wird meistens auf einer Grafikkarte ausgeführt (vgl. Shader, online).

Vertex

Ein Vertex (Plural Vertices) beschreibt die Eckpunkte von einem zwei- oder dreidimensionalen Polygon Mesh. Neben Position können in ihnen unter anderem auch Daten wie Farbe (siehe Abb. 15), Texturkoordinaten oder auch der Normalvektor gespeichert werden (siehe Abb. 16) (vgl. Vertex [computer graphics], online).

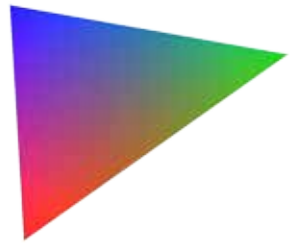


Abb. 15: Die Farben werden in den Vertices gespeichtert und die Bereiche dazwischen werden interpoliert

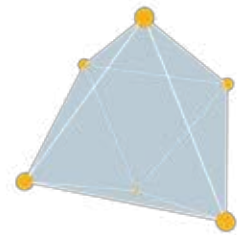


Abb. 16: Vertices eines Oktaeder

Vertexshader

Der Vertexshader läuft vor dem Pixelshader und wird ein Mal pro Vertex ausgeführt. Primär rechnet er für die Darstellung am Bildschirm die 3D-Koordinaten in 2D-Koordinaten um und gibt diese Daten an den Pixelshader weiter. Des weiteren können die Vertex-Daten dazwischen auch beliebig verschoben oder verändert werden (siehe Abb. 17) (vgl. Shader, online).

Pixelshader

Im Pixelshader wird die auszugebende Farbe berechnet. Diese kann von der Vertex-Farbe, Beleuchtung, Bump Mapping, Farbe aus einer Textur oder komplett anderen Quellen oder Effekten stammen und kann im Pixelshader unterschiedlich weiter verarbeitet werden (siehe Abb. 18) (vgl. Shader, online).

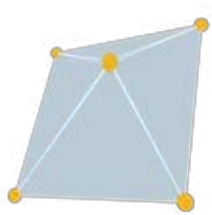


Abb. 17: Vertices werden im Vertex Shader verschoben

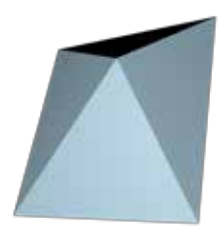


Abb. 18: Die auszugebende Farbe wird im Pixel Shader berechnet

Textur Koordinaten

Um dem Polygon Mesh mehr Details oder Farbe zu geben, werden Texturen verwendet (vgl. Texture mapping, online). Um jedoch ein zweidimensionales Bild auf einen dreidimensionalen Mesh zu legen, wird dieser Mesh aufgefaltet. Auf den flachen, aufgefalteten Mesh können dann Texturen gelegt werden. Die Punkte dieses flachen Meshes werden dann in den Texturkoordinaten mit dem Mesh gespeichert (vgl. UV mapping, online).

Die Texturkoordinaten sind üblicherweise Werte zwischen 0 und 1. Im Shader können auch Werte außerhalb dieses Bereichs benutzt werden. Dieses Verhalten kann pro Textur in beiden Achsen separat gesetzt werden (siehe Abb. 19) (vgl. Microsoft Developer Network 11.6.2013, online). Somit kann man, um Speicher zu sparen, nur die Hälfte einer Textur in einem Bild speichern, und die andere dann im Shader durch eine Spiegelung ergänzen (siehe Abb. 20).

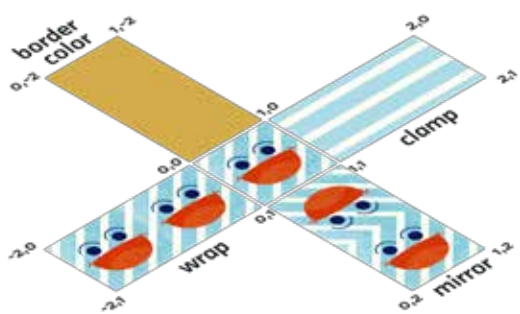


Abb. 19: Texturkoordinaten und ihr Verhalten mit unterschiedlichen Addressing Modes

Abb. 20: Alle Bilder für die Strecke befinden sich in unterschiedlichen Kanälen einer gemeinsamen Textur

Fahrzeug/Bike

Das Bike wurde in Cinema 4D gemodelt und animiert. Zusätzlich zur Datei fürs Geradeausfahren, gibt es noch zusätzliche Dateien, in denen sich das Model in weiteren Positionen wie Unfall, Lenken (siehe Abb. 21) und Springen befindet. Im Vertexshader wird dann linear zwischen den Positionen, Rotationen und Skalierungen dieser Dateien interpoliert. Die Farben des Bikes wurden in Maya in den Vertex-Farben gespeichert, da dies von Cinema 4D nicht unterstützt wird.

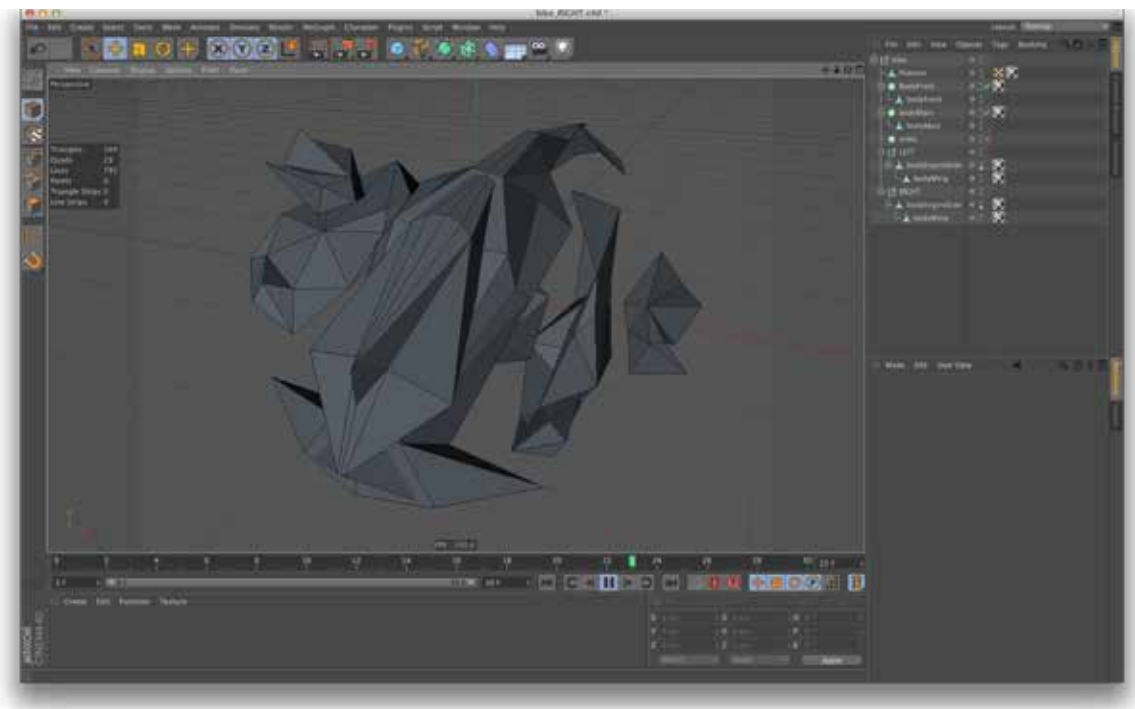


Abb. 21: Bike fährt nach links in Cinema 4D

Strecke und Landschaft

Rennstrecke und Landschaft wurden in Autodesk Maya erstellt. Dies erlaubt unter anderem den Export von Vertex-Farben und mehreren Texturkoordinaten. Außerdem ließ sich fast alles durch Skripts automatisieren. Damit konnten bestimmte Daten, wie z.B. die relative Position auf der Strecke, als Farbe eines Vertex gespeichert werden (siehe Abb. 22).

Im Vertexshader werden diese Daten ausgelesen, der dann die Hindernisse und Sprungrampen aufrichtet. Der Pixelshader zeichnet dann zum Beispiel die Pfeile auf die Sprungrampen und Ausrufezeichen auf die Hindernisse (siehe Abb. 23).

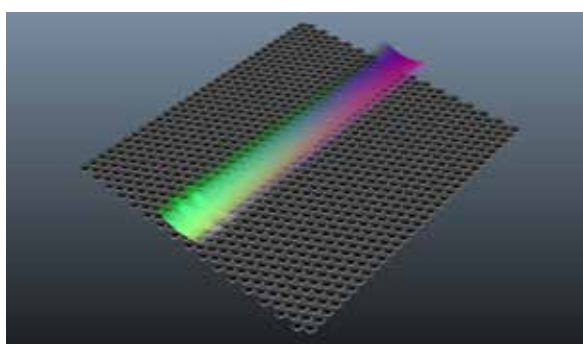


Abb. 22: Strecke und Landschaft in Maya, mit Normalvektoren und Vertex Farben

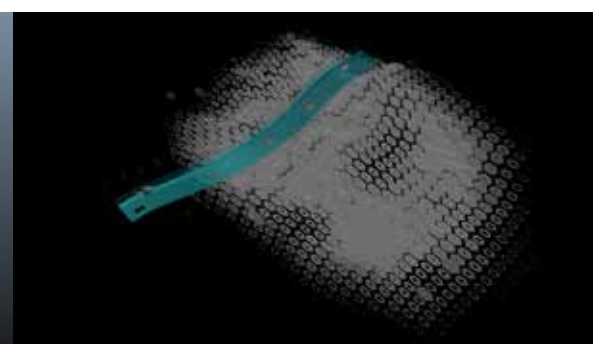


Abb. 23: Strecke und Landschaft in vvvv, nach dem Shader

Strecken- und Landschaftsbewegung

Praktisch alles, was in dem Spiel nach einer kontinuierlichen Bewegung aussieht, ist vorgetäuscht. Während des Rennens bewegt sich das Bike nie vorwärts.

Die einzelnen Vertices werden nur langsam nach vorne bewegt und springen wieder nach hinten, knapp bevor sie an die Stelle des nächstgelegenen vorderen Vertices kommen würden. Daten, wie die Bewegung nach oben, oder zur Seite, werden dann auch zur nächsten Vertex Reihe weitergeschoben (siehe Abb. 24).

Dadurch ergibt sich der Anschein einer kontinuierlichen Bewegung. Man muss hier allerdings darauf achten, dass die Enden der Strecke nicht hin und her zittern.

Die Daten, wie Hindernis- und Sprungrampen-Positionen, Verschieben nach oben/unten und Sinus und Cosinus für die

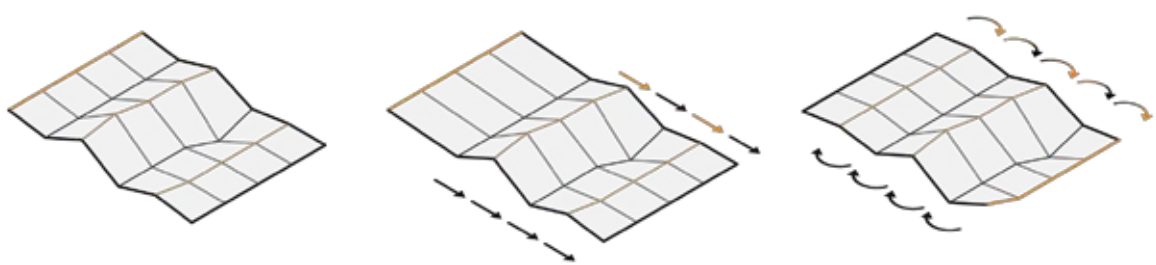


Abb. 24: Weitehrschieben von Vertices

großen Neigungen der Strecke werden mit einer pseudozufälligen Funktion generiert, in Warteschlangen gespeichert und dann, über mehrere Texturen, in den Shader geschickt.

Im Gegensatz zur Strecke, wo Vertices nur entlang von orthogonalen Achsen verschoben werden, wurden bei den Hexagons die Richtungen auf die Mitte zu und von der Mitte weg im Normalvektor gespeichtert, welcher sonst die Ausrichtung einer Fläche enthält. Wenn man die Vertices nun entlang dieses Vektors bewegt, kann man das Hexagon korrekt größer und kleiner machen, ohne die Relationen zwischen den Vertices zu kennen. Ähnlich kann man die Hexagons nach oben wachsen lassen, da die Vertex-Farbe die Information enthält, ob der Vertex sich unten oder oben befindet und ob dieser dementsprechend weiter nach oben bewegt werden sollte (siehe Abb. 25).

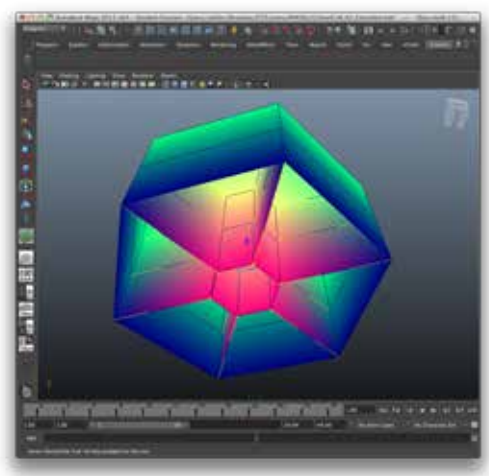


Abb. 25: Ein Teil der Landschaft von unten betrachtet: zu sehen sind die unterschiedlichen Vertex Farben und die gelben Normalvektoren

Bike und Kamera Position

Die Kameraposition hängt von der Position des Bikes in der Welt ab. Da die Strecke aber erst im Shader ihre endgültige Form bekommt, kann man in vvvv nicht direkt auf die Position der Vertices zugreifen, zumindest mit der im vorliegenden Projekt verwendeten Technologie.

Abb. 26: Die Quads, deren Positionen ausgelesen werden

Um das Nachprogrammieren des Vertexshader Codes in vvvv zu vermeiden, wurde der Streckencode erweitert. Er läuft ein zweites Mal mit einem vereinfachten Mesh, der nur die Reihe, auf der sich das Bike befindet, und zwei Reihen davor, damit das Bike auch richtig, nämlich der Strecke folgend, ausgerichtet ist, beinhaltet. Anstatt der Rennstrecke wird dann die Position des Vertex in der Farbe ausgegeben.

Abb. 27: Der Quad unter dem Bike ergibt die Position

Abb. 28: Die vier anderen Position ergeben jeweils 2 Vektoren

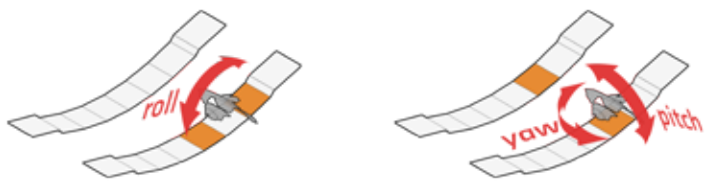


Abb. 29: Aus denen sich die Rotationen für das Bike ergeben

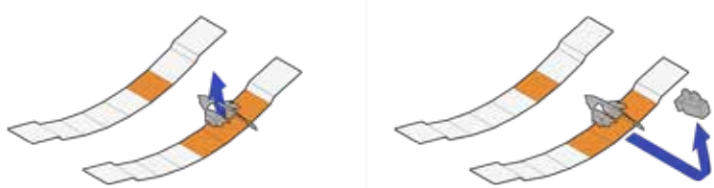


Abb. 30: Das Kreuzprodukt der Vektoren ergibt den Normalvektor, entlang dessen das Bike und weiter hinten die Kamera nach oben bewegt werden

Post Effects

Die Post-processing Effekte basieren auf überarbeitetem Code von Ylilammi, Jerry (2013, S.13) – 45 Grad Blur, Glow – und Tarlier, Francois (20.11.2009, ONLINE) – Lens Distortion.

Abb. 31: Ohne Post Effects

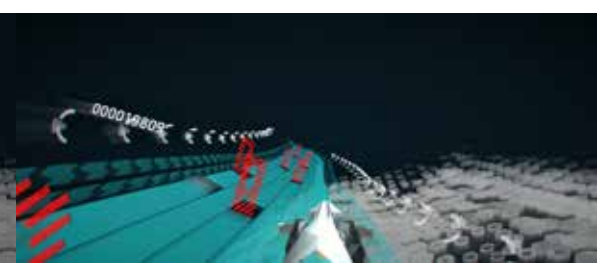


Abb. 32: Mit Post Effects

Blur, Glow

Der 45 Grad Blur Effekt entsteht, indem die Farben an mehreren Stellen in unterschiedlichen Distanzen und 90 Grad Inkrementen ausgelesen werden. Die Intensität einer einzelnen Farbe wird mit zunehmender Distanz schwächer (siehe Abb. 33).

Zusätzlich gibt es noch einen Radialen Blur. Hier werden, im Gegensatz zum 45 Grad Blur, zusätzliche Farben in Richtung Bildschirmmitte ausgelesen, was mit der verwendeten Kamera einen Motion-Blur-ähnlichen Effekt erzeugt (siehe Abb. 34).

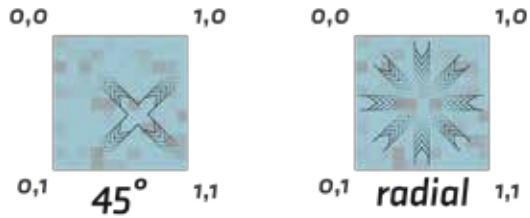


Abb. 33: Für den Blur und Glow wird für einen Pixel, von mehreren Stellen, Farbe ausgelesen und unterschiedlich stark dazu addiert

Abb. 34: Effekt auf Demo Textur

Vignette

Die Vignetteten Stärke ergibt sich aus der Distanz von der Texturkoordinate zu der Texturmitte. Da die Texturkoordinaten quadratisch sind und der Aspect Ratio der Ausgabe weiter ist, ergibt sich eine Ellipse (siehe Abb. 35). Die Vignette steuert ob die Pixelfarbe mit 1 oder mit sich selbst, ähnlich dem Multiply Blend Mode von Photoshop, multipliziert wird, was zu einer dunkleren Farbe führt (siehe Abb. 36).

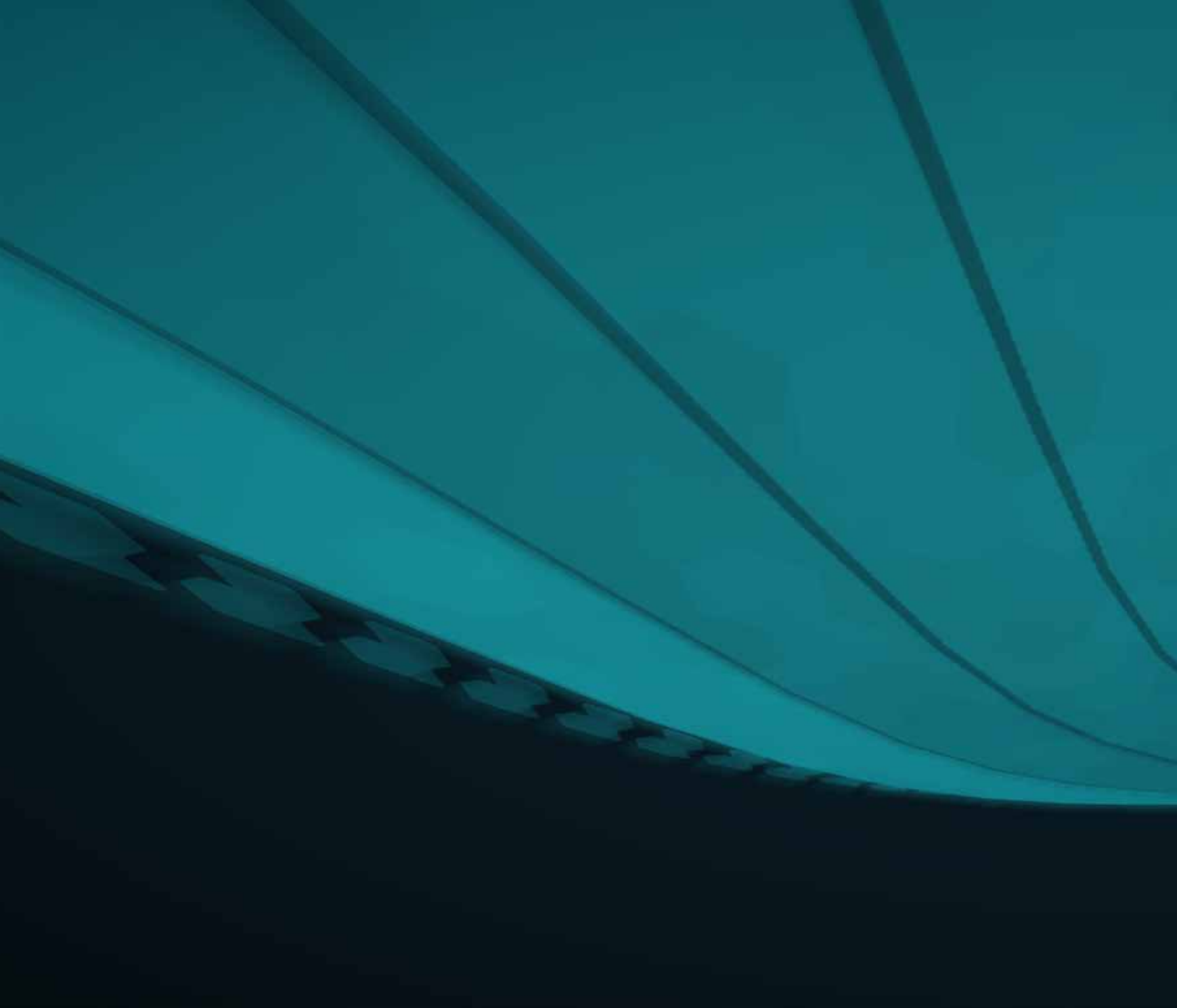
Abb. 35: Ellipsen förmige Vignette entsteht durch unterschiedliche Aspect Ratios

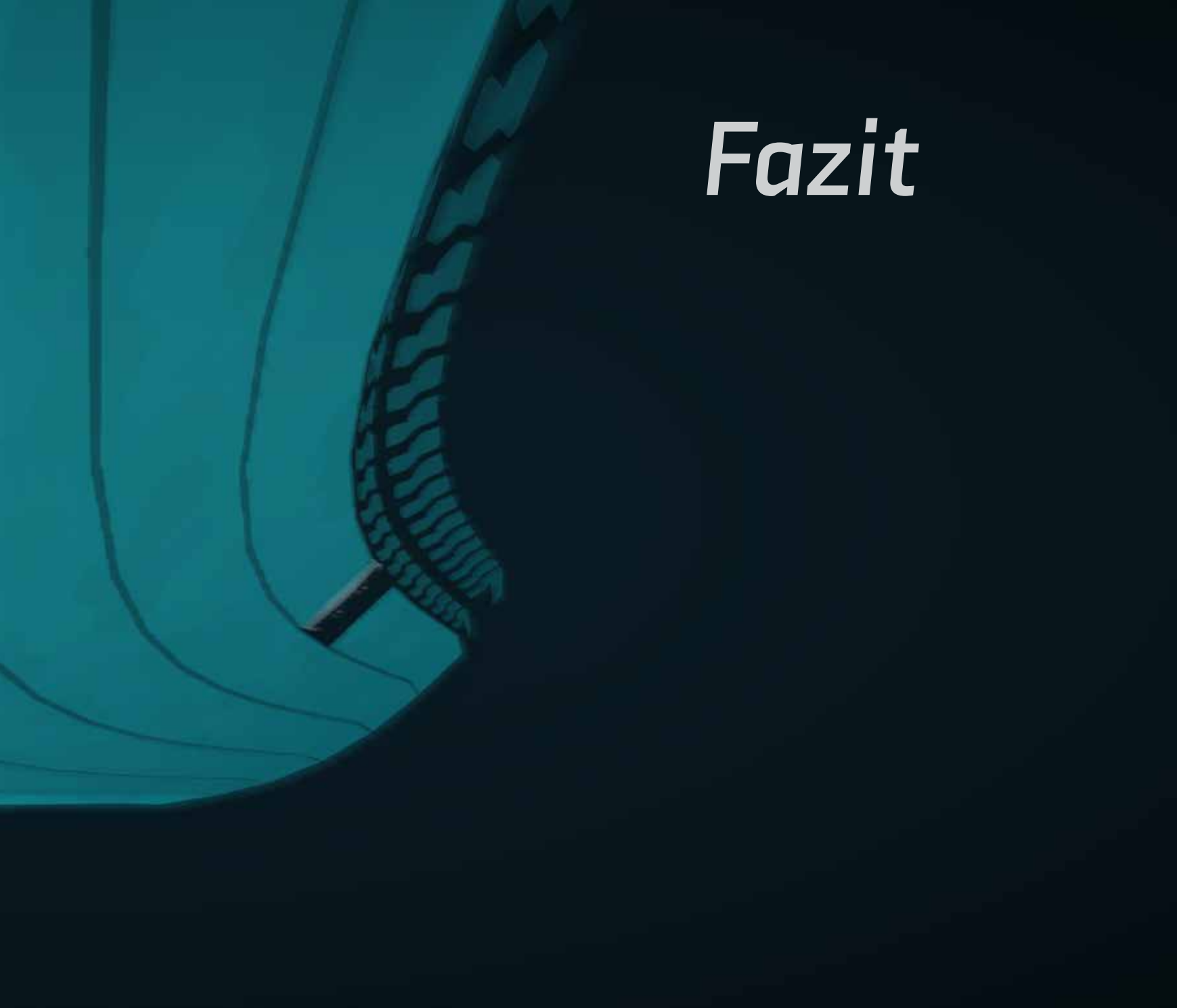
Abb. 36: Effekt auf Demo Textur

Chromatische Aberration

Die kissenförmige Verzeichnung zeigt den vollen Effekt beim roten Kanal. Die Texturkoordinaten für den blauen und grünen Kanal hingegen werden interpoliert zwischen den unveränderten Koordinaten und denen der kissenförmigen Verzeichnung. Dadurch ergibt sich ein anaglyph ähnlicher Effekt (siehe Abb. 37).

Abb. 37: Effekt auf Demo Textur





Fazit

Mit dieser Arbeit sollte gezeigt werden wie das Spiel entstanden ist und welche Veränderungen in einer neuen Version warum und wie durchgeführt wurden.

Betreffend die Spielmechanik wurden ausbremsende Teile am Spielende komplett entfernt. Dies betrifft etwa den Game Over Screen und das Unterschreiben des eigenen Highscores. Desweiteren wurden Erweiterungen innerhalb des Spielverlaufs hinzugefügt, wie Bonuspunkte für knappes und längeres Vorbeifahren an Hindernissen, um das Spielerlebnis interessanter zu gestalten.

Der optische Teil des Spiels wurde komplett neu geschrieben und verbessert. Vor dem Beginn dieser Arbeit hatte ich nur ein sehr geringes Verständis von Shadern und lernte dazu, sobald ich etwas brauchte. Beispielsweise war die Landschaft in einem frühen Versuch nur eine verformte Fläche, deren Hexagonmuster durch eine Textur entstand. Da ich so aber schnell an Grenzen stieß, schlug mein Bruder vor, anstelle der Textur die Hexagons als Meshes zu rendern. Hierfür brauchte ich dann Vertex-Farben, was von Autodesk Maya unterstützt wird, also lernte ich die benötigten Teile des Programms und hatte nach ein paar Tagen Ergebnisse.

Ähnlich ging es mit den Post Effekten. Online finden sich recht gut entsprechende Dokumentation und Code, der sich mit ein wenig Arbeit schnell verwenden lässt und zu eigenen Anpassungen anregt. Selbst wenn man nur selbstständig herumprobiert ist es zufriedenstellend, da

man schnell Ergebnisse sieht und direkte Kontrolle über die einzelnen Farben jedes Pixels hat.

Damit ergab sich ein futuristisches Racing Game mit Vollkörper-Einsatz, das durch fortgeschrittene Programmier- und Prototyping-Techniken verbessert wurde.

Literaturverzeichnis

Microsoft Developer Network (11.6.2013): Texture Addressing Modes (Direct3D 9). In: http://msdn.microsoft.com/en-us/library/bb206239(v=vs.85).aspx (zuletzt aufgerufen am 03.12.2010)

Shader. In: Wikipedia. The Free Encyclopedia. Stand: 26.6.2013, http://en.wikipedia.org/w/index.php?title=Shader&oldid=561625130 (zuletzt aufgerufen am 29.8.2013)

Tarlier, Francois (20.11.2009): GCubic Lens Distortion Shader. In: Francois Tarlier's blog, http://www.francois-tarlier.com/blog/cubic-lens-distortion-shader/ (zuletzt aufgerufen am 28.8.2013)

Texture mapping. In: Wikipedia. The Free Encyclopedia. Stand: 15.3.2013, http://en.wikipedia.org/w/index.php?title=Texture_mapping&oldid=544277745 (zuletzt aufgerufen am 31.8.2013)

UV mapping. In: Wikipedia. The Free Encyclopedia. Stand: 31.8.2013, http://en.wikipedia.org/w/index.php?title=UV_mapping&oldid=570893682 (zuletzt aufgerufen am 31.8.2013)

Vertex (computer graphics). In: Wikipedia. The Free Encyclopedia. Stand: 21.3.2013, http://en.wikipedia.org/w/index.php?title=Vertex_(computer_graphics)&oldid=546124232 (zuletzt aufgerufen am 29.8.2013)

vvvv group: Propaganda. In: http://vvvv.org/propaganda (zuletzt aufgerufen am 28.8.2013)

Ylilammi, Jerry: Making of Highway 4k. In: Jerry Ylilammi, http://ylilammi.com/webgl/highway4k/Making%20of%20Highway%204k.pdf (zuletzt aufgerufen am 28.8.2013)

Bildnachweis


© Copyright 2013 Stefan Kernjak IND 10