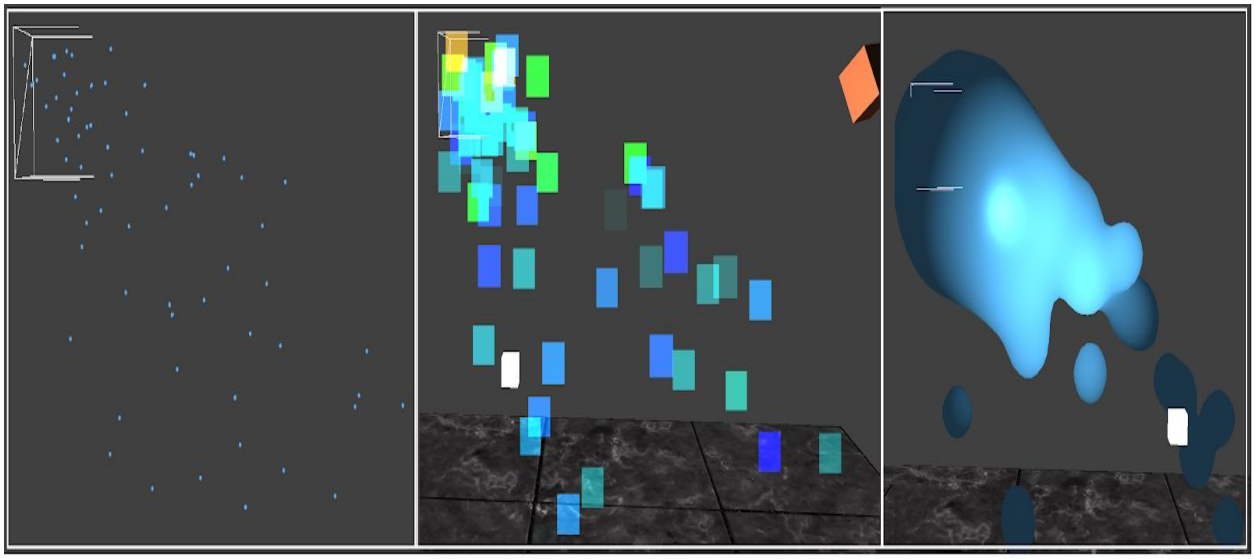

Dokumentation

Procedural Particle System

Real-time Rendering

28.02.2018

Valentin Kircher Bautista
Hochschule Trier
54293 Trier

Übersicht

Das Programm ist mit Win32 auf x86 Architektur programmiert und wurde für x64 nicht getestet. Um ein laufenden Programm zu haben, achten sie auf der Aktuell Zu Ausführenden version.

Die Szene

Die Szene (siehe Abbildung 1) besteht aus 5 Elementen. Ein Boden mit einer Textur, eine kleine Lampe (weißes Rechteck), die sich von links nach rechts bewegt, ein Cube (braunes Quadrat), der dazu dient die Beleuchtung zu testen, ein Partikelemitter (weiße Linien, die ein Quadrat bilden), dessen Normale auch visualisiert wurde und das Wichtigste von allem: die Partikel. Diese werden im Bereich des Partikelemitters auf prozedurale Weise erzeugt. Die Partikel können drei unterschiedliche Zustände annehmen. Mittels der Tastatur können diese drei Zustände angezeigt werden (siehe Abschnitt Keyboard Command).

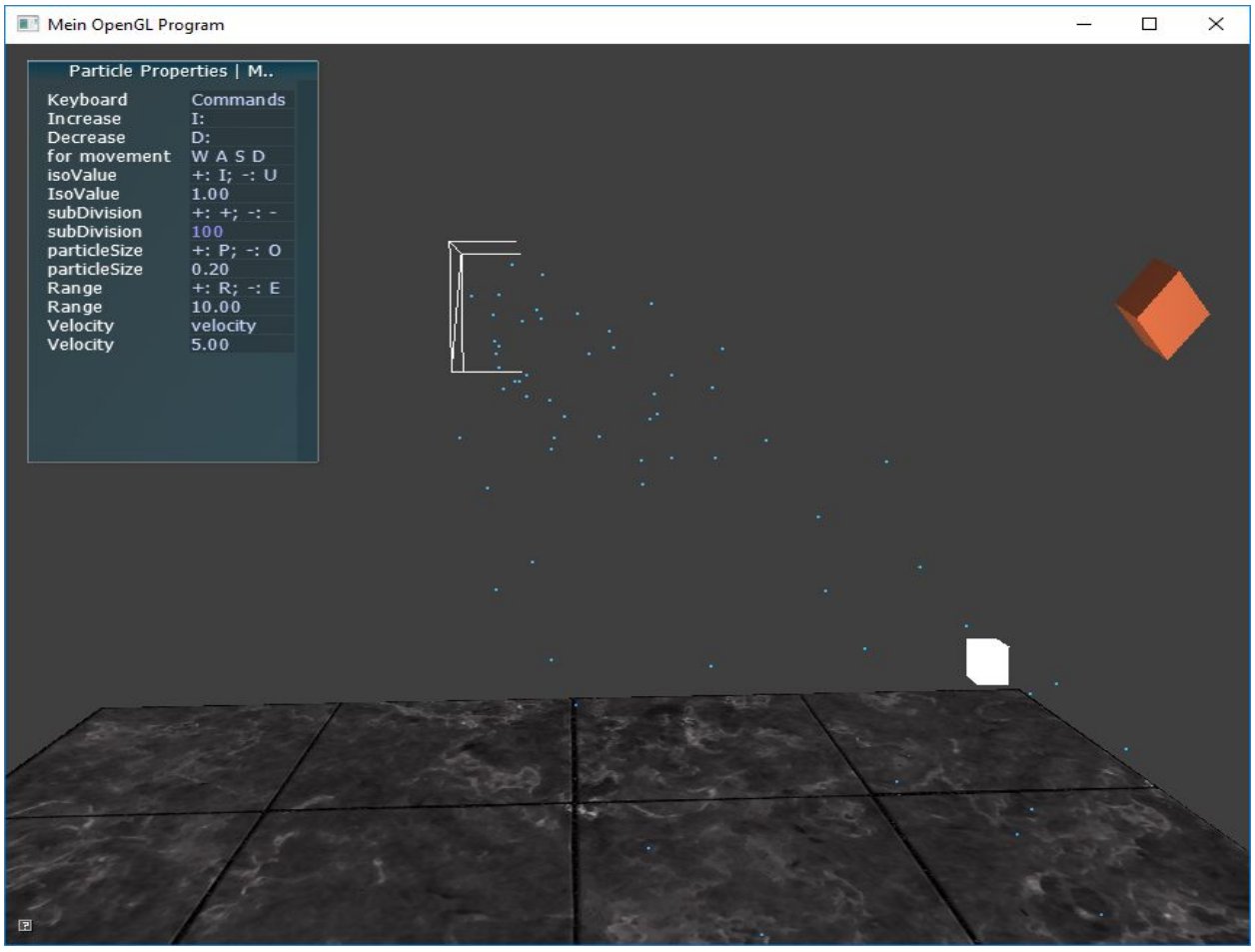

Abbildung 1. Darstellung der Szene

Die erste und einfachste Form Partikel zu visualisieren ist, sie als Punkte darzustellen. Zweitens können Partikel auch als Billboards abgebildet werden. Die spannendste Form der Darstellung von Partikeln sind die sogenannten Metaballs. Bei dieser Darstellungsform formen die Partikel eine implizite Fläche und somit wird eine Flüssigkeit abgebildet.

Anpassungsmöglichkeiten

Über die Tastatur können nicht nur die einzelnen Zustände abgerufen werden, sondern auch die Eigenschaften der Partikel und Metaballs können über die Tastatur verändert werden. So kann zum Beispiel die Größe der Partikel nach Belieben angepasst werden, als auch die Geschwindigkeit mit der sich die Partikel bewegen. Die einzelnen Funktionen werden in Abschnitt Keyboard Command aufgelistet.

Files

Das Projekt besteht aus vier .cpp Dateien mit zusätzlich 7 Headers.

- Die Hauptdatei partikelsystemvs.cpp beinhaltet jeweils für die Partikel und Metaballs die Parameter, als auch die Fenstererzeugung mit Hilfe von glfw.
- Für die GUI wird AntTweakbar benutzt.

Das Projekt verwendet C++ als Rahmenprogramm und OpenGL mit Vertex-, Geometry-, und Fragment Shader um den Flüssigkeitseffekt darzustellen.

Ziele

1. Umsetzung eines prozeduralen Fluid-Effects, ähnlich der Flüssigkeiten in dem Spiel Portal 2.
2. Die Partikel sollen in drei Formen abgebildet werden: Punkte, Billboards und Metaballs

Spezifikationen

Die **Header Dateien**: Renderer, Shader, Shader2, Shaderprogramm und die cpp Datei Shaderprogramm sind verschiedene Abstraktionen um die Shader Dateien einlesen zu können. Sie teilen OpenGL Funktionen für die Shader “Erzeugung”, “Compilation”, “Linkung” und “Uniform einlesen” in Methoden auf. Zusätzlich wird jeweils für die verschiedenen Verarbeitungsschritte eine Fehlererkennungsmethode erzeugt. Dadurch ist es einfacher in den verschiedenen Klassen Variationen zwischen den Shadern zu erstellen und zu testen.

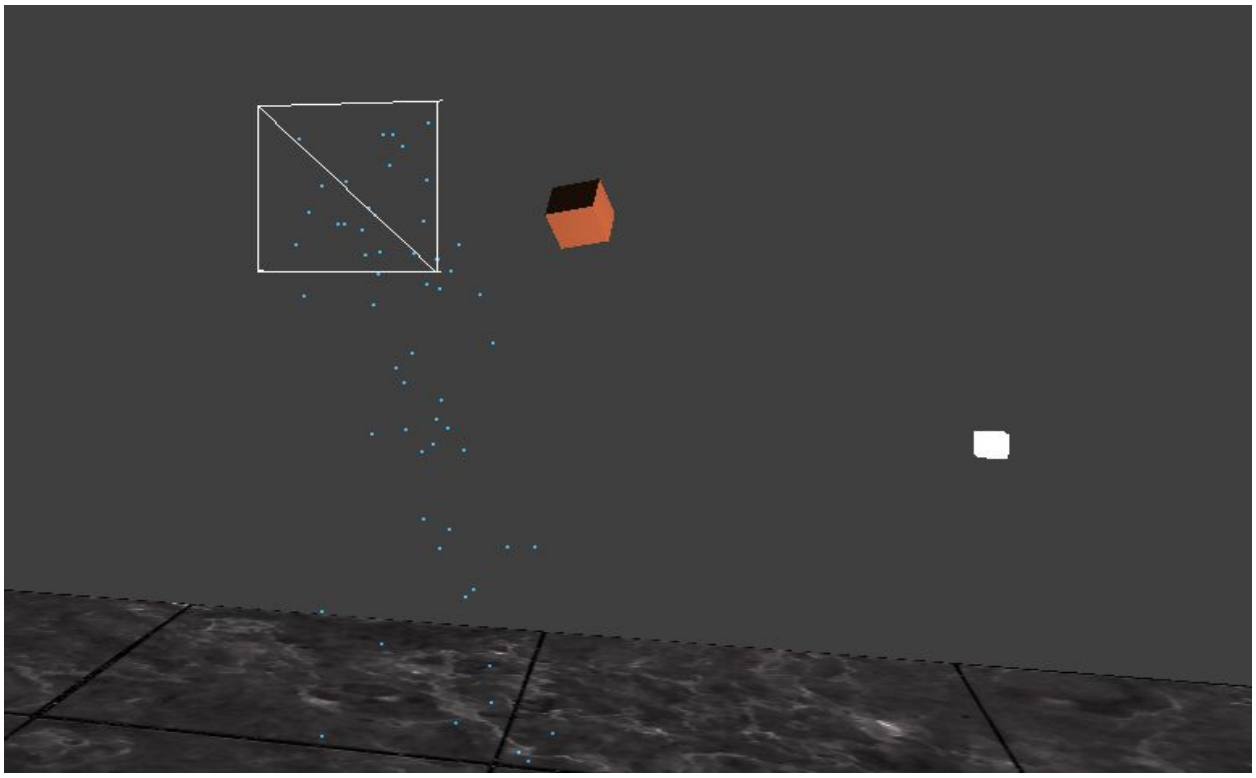

Abbildung 2. Kameraansicht des Emitters von der Seite mit Zoom

Die **Header Datei "Kamera"** beinhaltet verschiedene Methoden, damit die Kamera mit Keyboard Command eingelesen werden kann. Dadurch ist es möglich, sich durch den Raum zu bewegen und mit der Maus die Neigung der Kamera zu verändern. Des Weiteren ist es möglich mit dem Mausrad hinein und heraus zu zoomen. Dafür wurden der Klasse Parameter für die Position und drei Vektoren zugeteilt, wovon der Erste nach oben zeigt, ein Zweiter nach rechts und ein Dritter nach vorne (der Winkel zwischen jedem Vektor beträgt dabei 90°).

Abbildung 3.

Partikel-Einstellungen werden in real-time angepasst.

Die Header Datei **“Particle”** und die cpp “Particle” beinhalten alle Informationen, die benötigt werden, um die Partikel als Punkte, Billboards und Metaballs abzubilden. Ein Struct beinhaltet die verschiedenen Eigenschaften, die die Partikel besitzen sollen. Diese sind Position, Farbe, Größe, Gewicht, Geschwindigkeit, Lebensdauer und Typ. Die Variable Typ ist eine Integer Variable. Typ beschreibt den Zustand in dem sich der Partikel befindet. Es gibt zwei Formen des Typs. Wird der Typ auf 0 gesetzt, dann werden die Partikel generiert. Beim Wert 1 werden die Partikel gerendert. Aus dieser Information stellt der Konstruktor die Speicher-Allokationen auf der CPU für die verschiedenen Attribute bereit; jeweils mit einem Offset, um auf eine neue Stelle zu gelangen. Hier werden die Vertex-Array-Objekte und Vertex-Buffer-Objekte und für jeden Partikel die Parameter initialisiert. Für die Generierung der Partikel ist kein Fragment Shader notwendig, da diese nicht gerendert werden sollen. Daher ist nur die Zuweisung vom Vertex und Geometry Shader nötig.

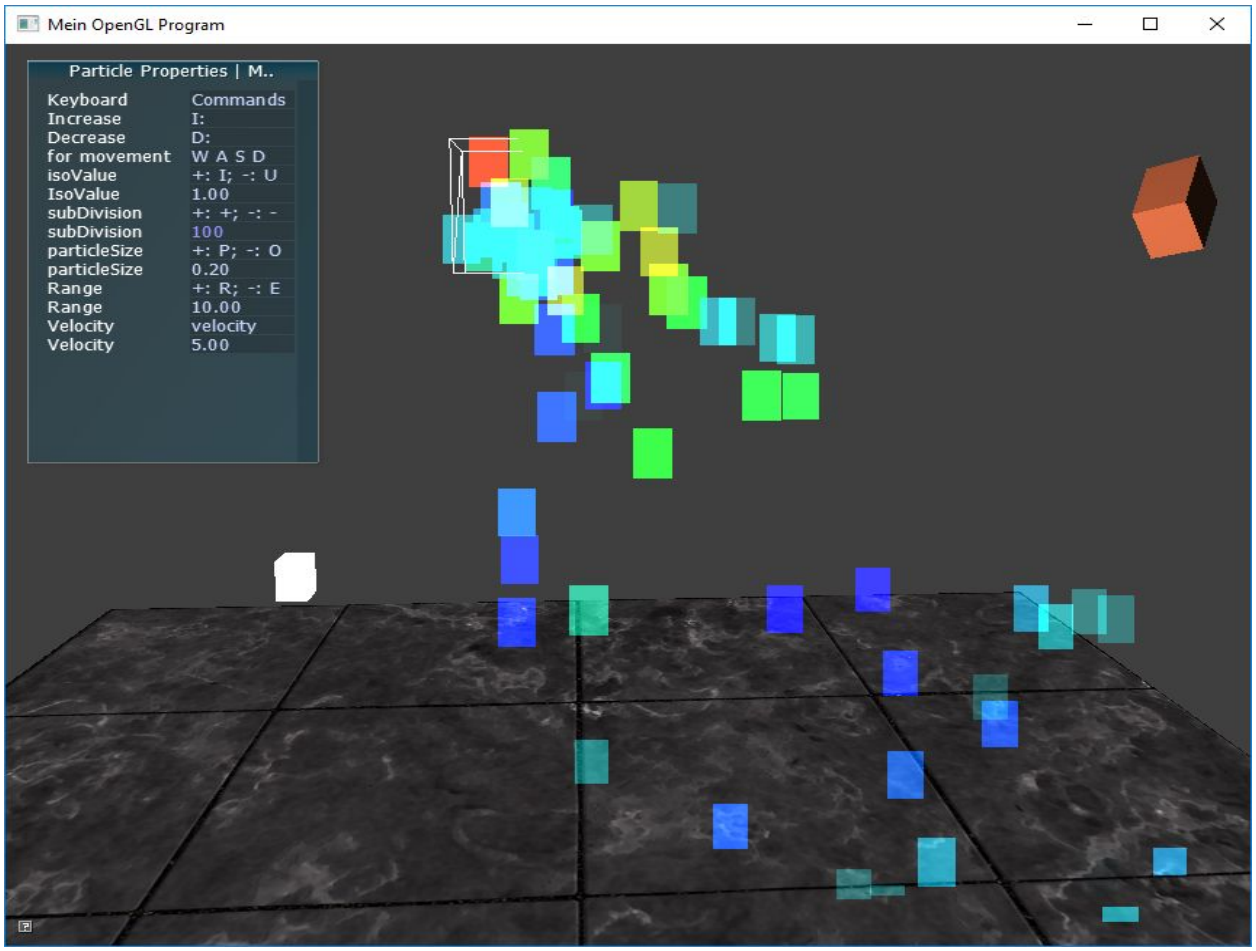

Abbildung 4. Anzeige der Partikel als quadratische Billboards.

Für das Rendern der Partikeln werden pro Zustand jeweils drei Shader eingelesen mit verschiedenen Variationen dazwischen.

Rendern und Buffer

In der Methode *Update particles* ist die Generierung der Partikel festgehalten. An dieser Stelle werden allen Uniform-Variablen lokale Werte zugewiesen. Solange die Zeit nicht abgelaufen ist, werden mittels der Rendermethode immer wieder neue zufällige Werte zugewiesen. Um die Partikel auf den Bildschirm anzuzeigen, müssen zwei Schritte erfolgen: Zuerst müssen die Partikel geupdatet und gerendert werden. Anschließend müssen bereits generierte Partikel “verfolgt” werden. Um diese “Verfolgung” kümmert sich das Transform Feedback. Es erfasst die ausgegebenen Geometrien. Transform Feedback ist somit der Prozess, der dafür sorgt, dass aus den generierten Partikeln die Primitiven gespeichert werden, um diese in der Zukunft wieder verwenden zu können. Im vorliegenden Fall wird jedes emittierte Partikel auf den zugewiesenen Buffer aufgezeichnet und dieser kann dann zum Rendern der Partikel verwendet werden.

Es muss auch, wie bereits erwähnt, eine doppelte Pufferung verwendet werden. Einerseits der Bufferserver als Lesepuffer und ein weiterer Buffer als Schreibpuffer. Des Weiteren muss der Tranformations- Feedback- Buffer initalisiert werden. Die gesamte Initialisierung des Partikelsystems, einschließlich der Initialisierung der Transformationsrückkopplung, erfolgt in der ziemlich langen Funktion "InitalizeParticleSystem ()". Der erste Teil beschäftigt sich mit Shader-Laden:

Neben dem Code zum Laden von Shadern gibt es nur eine Transform Feedbackfunktion: glTransformFeedbackVaryings. Dies ist die Funktion, die OpenGL mitteilt, welche Eckpunktattribute durch Transformationsrückmeldung aufgezeichnet werden sollen. In unserem Fall verwenden wir einfach alle Partikelattribute. Der erste Parameter ist die Shader-Programm-ID, der Zweite ist die Gesamtzahl der aufgezeichneten Attribute, der Dritte ist der String-Name einer Ausgangsvariablen, die aufgezeichnet werden soll (dieser Name entspricht der Ausgangsvariablen im Geometrieshader). Der letzte Parameter, der in unserem Fall eingesetzt wird, ist der GL_INTERLEAVED_ATTRIBS. Dieser wird genutzt, da unsere Ausgabe in einem einzelnen Buffer geschrieben wird und die Partikel nacheinander gespeichert werden.

Der zweite Teil von InitParticleSystem () beschäftigt sich mit allen notwendigen Buffern:

Die UpdateParticles () Funktion benötigt nur einen Parameter und zwar die seit dem letzten Frame vergangene Zeit. In erster Linie müssen alle Generator-Uniformen einstellt werden. Wann immer die Zeit, die lokal gezählt wird, eine bestimmte Schwelle erreicht (fNextGenerationTime), muss die Anzahl der gewünschten erzeugten Partikel und zufälligen Werte in dem Shader-Programm eingestellt werden, damit die Partikel richtig erzeugt werden können.

Bei der Generierung der Partikel werden die Partikel nur aktualisiert, das bedeutet, dass keine grafische Ausgabe gemacht wird und man somit keine Rasterisierung benötigt. Daher

kann diese deaktiviert werden. OpenGl wird in diesem Fall mitgeteilt, dass das zuvor erstellte Tranformations-Feedback-Bufferobjekt verwendet werden soll und dadurch wird der aktuelle VAO gebunden d.h. der Lesepuffer. Als nächsten Schritt wird nur das erste Attribut (die Geschwindigkeit) aktiviert, da beim Aktualisieren von Partikeln ein Geschwindigkeitsvektor benötigt wird. Beim Rendern wird jedoch keine Geschwindigkeit gebraucht. Auf diese Weise kann eine gewisse Verarbeitungszeit eingespart werden, da Dinge die nicht verwendet werden auch nicht gesendet werden.

In einem weitern Schritt wird OpenGL der Ort mitgeteilt, an dem es das Ergebnis der Transform-Feedback-Operation speichern soll. Der aktuelle Lesebuffer ist iReadBuffer und der Schreibbuffer nennt sich 1-iCurReadBuffer. Dies bedeutet, dass der VBO uParticleBuffer [1-iCurReadBuffer] als Speicher für die Transformationsrückmeldung dient.

An dieser Stelle ist alles für das Rendering mit Transformationsfeedback eingerichtet. Als nächstes wird die Anzahl der ausgegebenen Primitive gezählt. Anschließend wird OpenGL mitgeteilt, dass mit der Funktion glBeginTransformFeedback (GL_POINTS) das Feedback-Rendering transformieren werden soll. Danach wird die eigentlich Zeichenfunktion mit der Anzahl an Partikeln als Parameter aufgerufen und die Tranformationsrückmeldung beendet. Da das Rendern abgeschlossen ist, kann an dieser Stelle auch die Abfrage “glEndQuery” beendet werden.

Darstellung

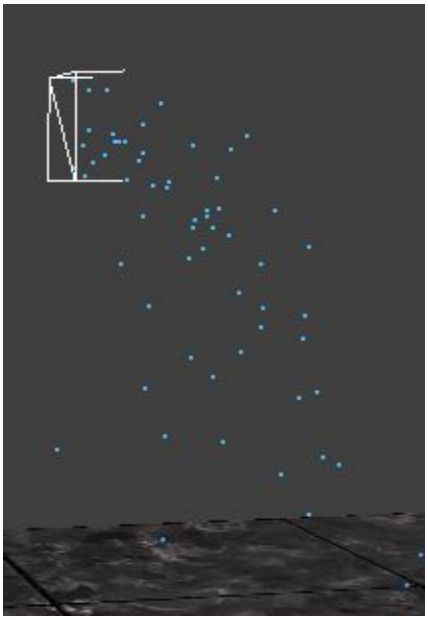

Abbildung 5. Partikel-Punkte “Fall”

aus dem Emitter

Punkte:

Die RenderParticles Methode benötigt zwei Parameter: “Metaball” und “LichtObject”.

Um die Partikel als Punkte darzustellen muss lediglich ein Shader Programm erstellt werden, das diese Partikle aufnimmt und rendert.

Zusätzlich werden alle Uniformsvariablen gesetzt, die von den Shadern Programmen benötigt werden.

Dieses benötigt keinen Geometry Shader, da die Punkte nicht in andere Formen verändert werden und für die Uniforms werden nur die beiden Matrizen verwendet.

Billboards:

In einem nächsten Schritt sollen die Partikel in Form von Billboards abgebildet werden. Billboards sind viereckige flache Ebenen, die immer auf die Kamera gerichtet sind. Dafür nehmen wir einfach den Ansichtsvektor der Kamera und berechnen dann die Vektoren der Billboard-Ebenen. Hierfür müssen zwei Vektoren ausfindig gemacht werden: vQuad1 und vQuad2:

Diese zwei Vektoren sollten zusammen mit dem Sichtvektor senkrecht zueinander sein. Wenn alle drei Vektoren senkrecht zueinander stehen, erzeugen sie eine orthonormale Basis. Ausfindig können diese Vektoren anhand der SetMatrices Methode gemacht werden.

Diese Funktion berechnet nicht nur diese zwei Werte, sondern berechnet auch die Projektions- und Viewmatrix für das Partikel-Render-Programm. Der erste View Vektor wird über das Kreuzprodukt des Ansichtsvektors und des Kamera-UP-Vektors berechnet und normalisiert. An dieser Stelle werden dann diese zwei Vektoren als Uniforme für das Rendering-Shader-Programm gesetzt.

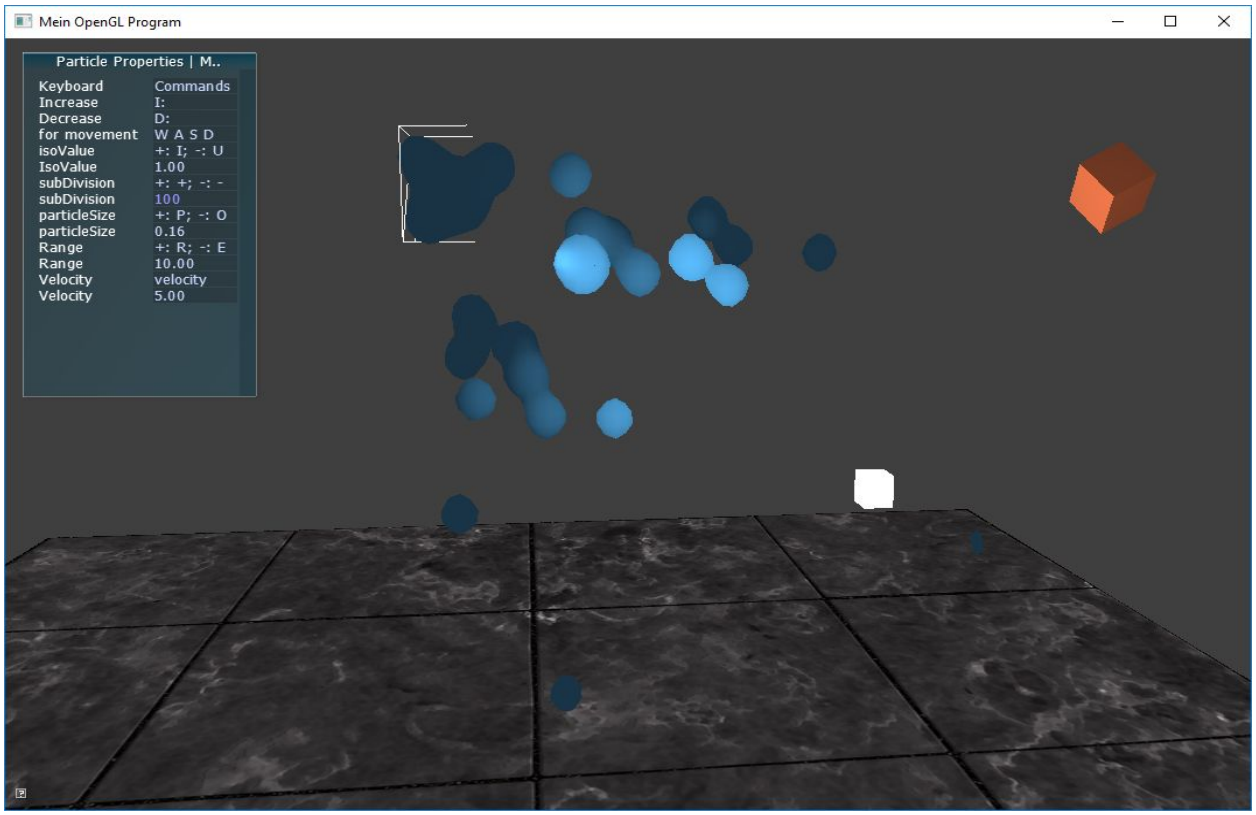

Abbildung 6. Partikel als Metaballs mittlerer Grösse

Das Rendering-Shader-Programms besteht aus Vertex-, Geometrie- und Fragment-Shader. Der Vertex-Shader leitet Daten weiter in den Geometry-Shader, dieser erstellt von einem Punkt ein Quad. D.h. es müssen aus einem Punkt 4 Ecken emittiert werden. Quads bestehen aus zwei Dreiecken und werden dadurch gebildet, indem 4 Ecken in der zuvor

definierten Reihenfolge ausgegeben werden. Die Gleichheitsvariablen vQuad1 und vQuad2 sind diejenigen, die schon in einem vorherigen Schritt berechnet wurden.

Metaballs (Partikel als Implizite Fläche):

Mit dem "Marching Cubes" Algorithmus von Paul Bourke (siehe http://paulbourke.net/geometry/polygonise/) können die Punkte auch eine implizite Fläche formen. Bei dieser Methode werden über den Geometry Shader aus einem Punkt acht weitere Punkte um einen Würfel herum erzeugt. In einem nächsten Schritt werden aus diesem Würfel ein Netz aus Dreiecken aufgebaut mit insgesamt 256 möglichen Verbindungen und ein zweites Netz für die Eckpunkte des Würfels erstellt. Diese zwei Netze sind in Tabellen auf der CPU gespeichert. Mit diesen Tabellen wird ein Netz gebildet. Aus den möglichen Verbindungen des Netzes wird ein Metaball auf Laufzeit in der Grafikkarte erzeugt. Sobald ein "Metaball" sich in einen anderen "Metaball" "hinein" bewegt, entsteht eine Intersektion aus diesen Metaballs. Dann werden die nächsten passendsten Dreiecke aus der Liste gewählt, um eine implizite Fläche zwischen den Metaballs zu bilden.

Der Fragment Shader für die Punkte benötigt nur zwei statische Attribute: Farbe und Größe. Die Position und Lebensdauer aktualisieren sich immer wieder. Die Billboards verändern ihre Farbe nach Lebensdauer von rot nach grün bis hellblau.

Die Metaballs benutzen phong-Shading als Beleuchtungssystem, mit weichem, diffusen und hartem (Spot)Licht.

Die Datei Particle.cpp enthält auch weitere Methoden. Dazu gehören die Methode zum Konfigurieren und Verändern der Eigenschaften und eine weitere Methode, um die Matrizen zu aktualisieren. Bei dem vorliegenden Projekt werden nur Projektion und viewmatrix benötigt, um eine zusätzliche Methode für die Kameraeigenschaft zu erstellen.

Die setRenderMode() Methode setzt die Partikel auf die verschiedenen Darstellungsformen.

Keyboard Commands

Allgemein gültige Tastaturbefehle

w, s, a, d	-> Kamerabewegung (nach vorne (zoom), nach hinten, links und rechts)
Mouse	-> Kamerabewegung (jaw, pitch, roll, zoom)
1	-> Partikel als Punkte darstellen
2	-> Partikel als Billboards darstellen
3	-> Partikel als Metaballs darstellen
0	-> Anzeige aller Elemente als Wireframe
f	-> "füllt" die Objekte wieder (von Wireframe zurück zur Normalansicht)
c	-> aktiviert die Kamera und blendet die Maus aus
Tabulator	-> zeigt die Maus außerhalb des Programmfensters und erlaubt dessen Anpassung
Esc	-> schließt die Anwendung

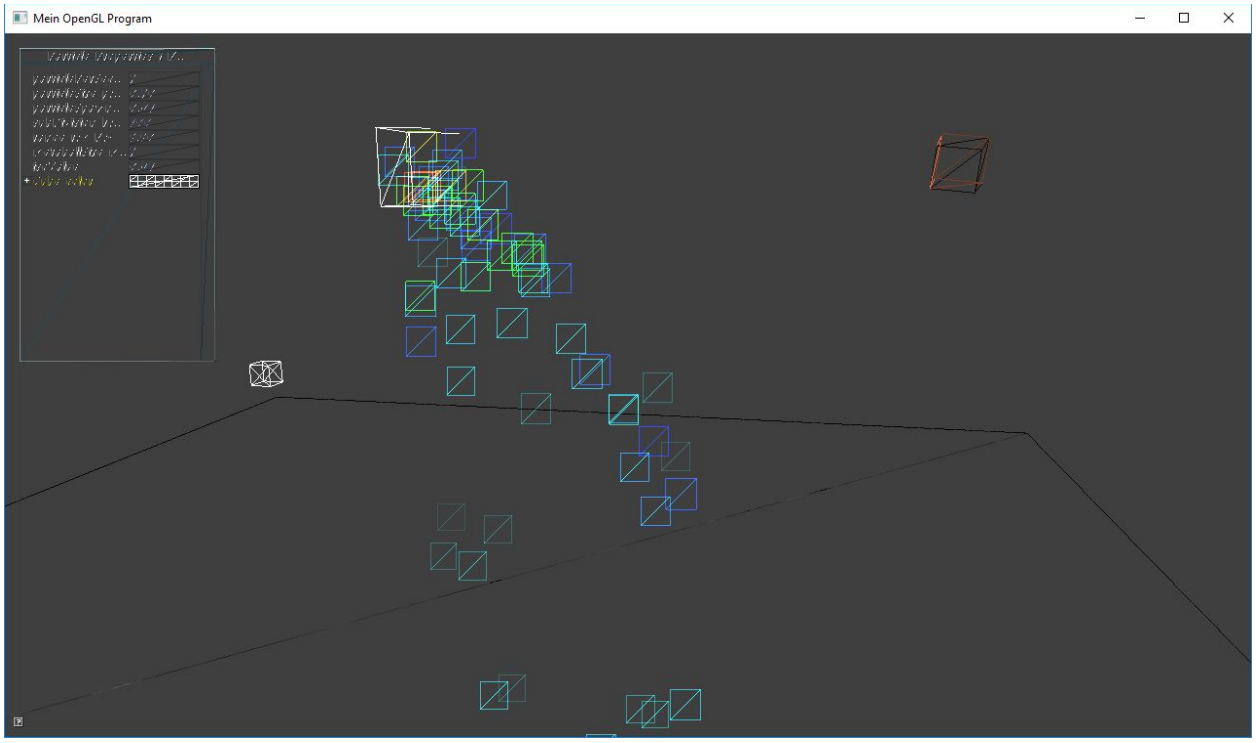

Abbildung 7. Anzeige aller Elemente als Wireframe

Steuerung von Metaballs (nach Eingabe von “3”)

n	-> Metaballs ausblenden
m	-> Metaballs einblenden
p	-> Metaballos verkleinern
o	-> Metaballs vergrössern
r	-> Range der Metaball erhöhen
e	-> Range der Metaball verkleinern
“+”	-> SubDivision der Metaball vergrößern
“-”	-> SubDivision der Metaball verkleinern
v	-> Geschwindigkeit erhöhen
b	-> Geschwindigkeit verringern
u	-> Iso Wert erhöhen -Lichtstrahl vergrössert sich
i	-> Iso Wert verringern-Lichtstrahl verkleinert sich

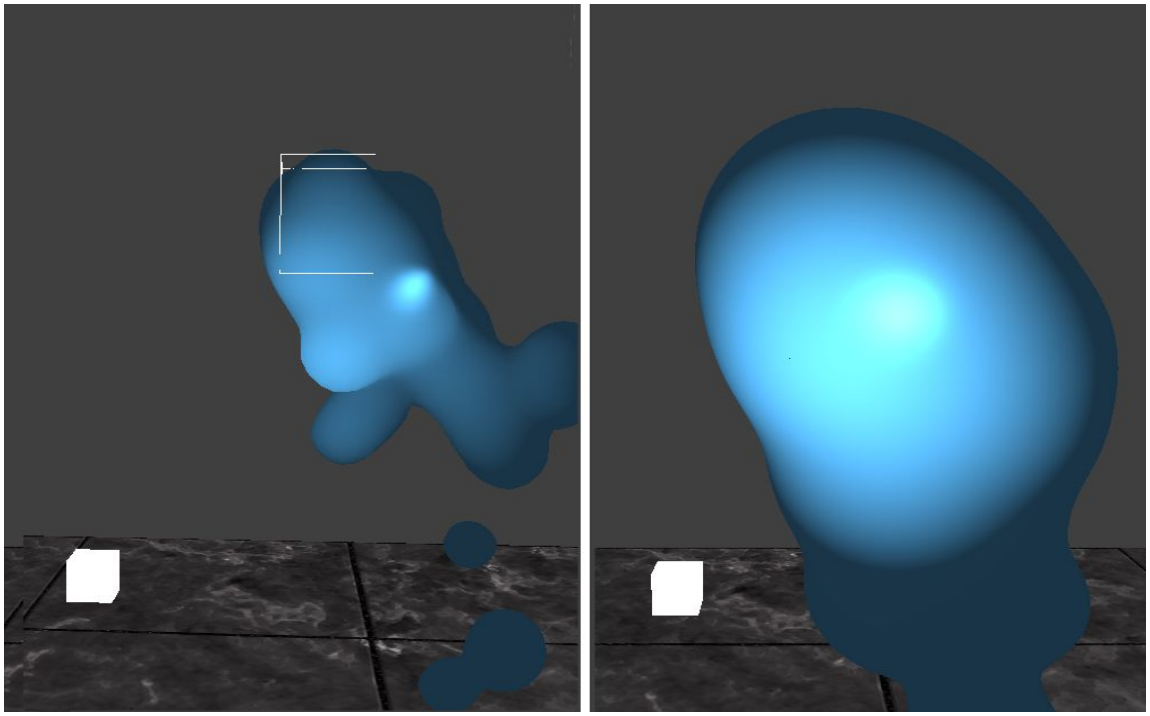

Abbildung 8. Metaballs mit geringem und hohem ISO-Wert