Institut für Visualisierung und Interaktive Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 146

Eine browserbasierte Entwicklungsumgebung für prozedurale Texturen

Christoph Kleine

Studiengang:	Informatik
Prüfer/in:	Jun.-Prof. Dr.-Ing. Martin Fuchs
Betreuer/in:	M.F.A. Lena Gieseke
Beginn am:	15. Mai 2014
Beendet am:	14. November 2014
CR-Nummer:	D.1.7, D.2.6

Kurzfassung

Texturen für digitale Spiele und Filme werden immer detailreicher und größer. Prozedurale Texturen können in beliebiger Auflösung hochkomplexe Muster und Strukturen erzeugen. Die Entwicklung von prozeduralen Texturen mit Kontrollparametern ist kompliziert. Zum einen sollen realistische, natürlich aussehende Muster erstellt werden, und zum anderen soll es möglich sein, Referenzbilder durch die passende Einstellung der Kontrollparameter nachzubilden. Um den Herstellungsprozess von prozeduralen Texturen zu vereinfachen, wird in dieser Arbeit eine plattformunabhängige, browserbasierte Entwicklungsumgebung für prozedurale Texturen vorgestellt. Diese erlaubt das Programmieren und Anzeigen von prozeduralen Texturen, die entweder in der Hochsprache Javascript oder in der Shader-Sprache geschrieben werden. Für die Texturen lassen sich Parameter definieren. Für diese werden von der Entwicklungsumgebung dynamisch „Slider“ erzeugt. Außerdem ist in die Entwicklungsumgebung ein Fenster, in das ein Referenzbild geladen werden kann, aus dem der Benutzer zum Beispiel Farben extrahieren kann, und die Funktionalität Benutzerinteraktionen aufzuzeichnen, integriert. Zusätzlich wird in dieser Arbeit ein Programmierbeispiel für prozedurale Texturen mit der Entwicklungsumgebung geboten und die Anforderungen an eine Studie diskutiert, die Rückschlüsse auf eine automatisierte Einstellung von Parametern einer prozeduralen Textur liefern kann.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

1 Einleitung

1.1 Motivation

Durch stetig wachsende Auflösungen von digitalen Anzeigegeräten und immer höhere Anforderungen an das finale Aussehen von computergenerierten 3D-Szenen wird das Gestalten von Objekt-Oberflächen immer wichtiger. Texturen spielen eine essentielle Rolle bei diesem Prozess, sie können sowohl die Farbgestaltung als auch Reflektionen und Verformungen von 3D-Objekten beeinflussen. Durch die höheren Anforderungen wird das Erstellen von Texturen für die Medienbranche immer schwieriger und aufwändiger. Digital erzeugte Oberflächen, deren Texturen zu grob oder zu ungenau sind, reichen aus, um einen Film oder ein digitales Spiel qualitativ herabzustufen und für den Kunden uninteressant zu machen.

Prozedurale Texturen bieten die Möglichkeit, komplexe Texturen in unbegrenzter Auflösung mit einer fast unendlichen Variation direkt auf der CPU oder sogar auf der GPU zu berechnen. So wird der Aufwand zur Speicherung von Texturen stark reduziert, da nicht hochauflösende Bilder, sondern Prozeduren gespeichert werden. Prozedurale Texturen sind flexibler als aus Bildern erzeugte Texturen und können sich beliebigen Strukturen anpassen. Ein Beispiel hierfür sind prozedural erzeugte Baumstamm-Texturen, die im Gegensatz zu Bild-Texturen, die nur eine endliche Datenmenge anbieten, in alle Dimensionen ausreichend berechnet werden können. Der Aufwand zum Anpassen einer Textur an ein Objekt sinkt deshalb.

Das Entwerfen von Textur-Programmen, die zu einem gewünschten Aussehen eine prozedurale Textur erstellen, ist sehr aufwändig. Ein wichtiger Aspekt dabei ist, dass die Texturen möglichst realistisch aussehen sollen. Eine zu auffällige Wiederholung wirkt unnatürlich. Ein Teil der prozeduralen Texturen benutzt deshalb zufällig generiertes Rauschen als Basis. Die Kontrolle des finalen Aussehens ist deshalb sehr schwierig. Kleine Änderungen im Textur-Code können außerdem große Änderungen bei der Textur erzeugen.

Um Texturen in der Grafikkarte zu berechnen, bietet sich WebGL zusammen mit der Programmiersprache OpenGL Shading Language an. Die für WebGL verfügbaren Shader können in zwei Arten unterteilt werden: Den Vertex-Shader, der für das Transformieren der Geometrie des Objekts ist, und den Fragment-Shader, der zum Beispiel für die Einfärbung der Pixel zuständig ist. Die beiden Webseiten Shadertoy [Sha] und GLSLSandbox [GLS] bieten Konzepte, die einen kleinen Teil der Anforderungen erfüllen, die an eine browserbasierte Entwicklungsumgebung gestellt werden. So wird die Möglichkeit geboten, Fragment-Shader zu programmieren und anzusehen. Außerdem wird eine große Datenbank von Shadern zum Testen bereitgestellt. Änderungen der Shader werden jeweils direkt im eingebetteten Editor vorgenommen. Als Eingabeparameter, um die Texturen zur Laufzeit zu verändern, werden die aktuelle Zeit oder auch Maus- und Tastatur-Eingaben angeboten.

1.2 Ziele dieser Arbeit

In dieser Arbeit wird eine browserbasierte Entwicklungsumgebung für prozedurale Texturen implementiert, die das Erzeugen von prozeduralen Texturen stark vereinfacht, indem sie den Fokus auf das Generieren von Prozeduren und Parametern setzt, und es dem Benutzer erlaubt, die prozeduralen Texturen „on the fly“ mit Hilfe der Parameter zu beeinflussen. Die Erzeugung des finalen Vertex- und des Fragment-Shaders wird von der Entwicklungsumgebung übernommen Zusätzlich werden automatisch Slider für das Verändern der Parameter und weitere Elemente bereitgestellt. Der Aufwand für den Benutzer sinkt, da er sich weder um das Kompilieren des Textur-Codes noch um die Erzeugung der UI-Elemente kümmern muss. Außerdem wird eine Interaktionsmöglichkeit mit einem Eingabebild geboten, die es erlaubt, Farben zu extrahieren und dadurch die Reproduktion dieses Bildes durch eine prozedurale Textur zu vereinfachen. Um weitere Features zu finden, wird ein Paper analysiert, welches das automatische Anpassen von prozeduralen Texturen an ein Referenzbild behandelt. Als letzter Teil wird eine Technik zum Tracking von Benutzer-Interaktionen implementiert. Die Idee dabei ist, Rückschlüsse auf eine automatische Angleichung einer prozeduralen Textur mit Parametern an ein Referenzbild zu ermöglichen.

Gliederung

Im ersten Teil dieser Bachelorarbeit wird eine browserbasierte Entwicklungsumgebung für prozedurale Texturen vorgestellt. Dazu werden zum einen analysiert, wie Texturen entwickelt werden und welche Methoden es dafür gibt und zum anderen, welche Elemente für die Entwicklungsumgebung wichtig sind. Zusätzlich werden Ablaufschritte aufgezählt, die zum Programmieren von prozeduralen Texturen mit der browserbasierten Entwicklungsumgebung notwendig sind.

Der zweite Teil der Bachelorarbeit beinhaltet den Transfer von Referenzbild-Eigenschaften auf eine prozedurale Textur mit Hilfe von Parametern.

Im letzten Teil wird analysiert, wie das Aufzeichnen von Benutzerinteraktionen die Qualität eines Programmes oder einer Webseite verbessern kann. Außerdem wird erklärt, welche Funktionen für das Aufzeichnen von Benutzerinteraktionen implementiert sind und wie eine Studie aussehen kann, die Rückschlüsse auf eine automatisierte Einstellung von Parametern einer prozeduralen Textur erlangt, sodass sich die Textur kaum von einem Referenzbild unterscheidet.

Am Ende folgt eine Zusammenfassung und ein Ausblick auf weitere Features und Möglichkeiten der browserbasierten Entwicklungsumgebung.

Kapitel 2 – Eine Entwicklungsumgebung: Beschreibung der browserbasierten Entwicklungsumgebung: Unter anderem mit der Analyse von verwandten Arbeiten, der Erklärung von notwendigen Grundlagen, sowie einem detaillierten Einblick in den Aufbau und die Funktion der Entwicklungsumgebung.

Kapitel 3 – Transfer von Referenzbild-Eigenschaften: Hier wird beschrieben, wie der Transfer von Referenzbild-Eigenschaften auf die prozedurale Textur funktioniert. Dazu wird auf verwandte Arbeiten eingegangen sowie kurz erklärt, welche Möglichkeiten die browserbasierte Entwicklungsumgebung dazu bietet.

Kapitel 4 – Aufzeichnen von Benutzer-Interaktionen: Beschrieben wird, wie das Aufzeichnen von Benutzer-Interaktionen bei Webseiten funktioniert und wie dadurch die Qualität verbessert werden kann. Es wird außerdem erklärt, wie das Aufzeichnen von Benutzerinteraktionen mit der browserbasierten Entwicklungsumgebung funktioniert und wie eine Studie aussehen kann, die versucht, Rückschlüsse auf die automatisierte Einstellung von Parametern einer prozeduralen Textur zu erlangen.

Kapitel 5 – Zusammenfassung und Ausblick: Hier wird eine kurze Zusammenfassung der Bachelorarbeit sowie ein Ausblick auf mögliche Features und Verbesserungen der browserbasierten Entwicklungsumgebung gegeben.

2 Eine Entwicklungsumgebung

In dieser Arbeit wurde eine browserbasierte Entwicklungsumgebung für prozedurale Texturen entwickelt, die den Prozess der prozeduralen Texturen-Entwicklung vereinfacht. Im Abschnitt Verwandte Arbeiten wird analysiert, wie Texturen entwickelt werden und welche browserbasierte Entwicklungsumgebungen es für prozedurale Texturen gibt. Im Abschnitt Grundlagen werden Themen behandelt, die dem besseren Verständnis von prozeduralen Texturen oder Elementen der Entwicklungsumgebung dienen. Im dritten Abschnitt wird schließlich erklärt, welche Anforderungen an die Entwicklungsumgebung gestellt und wie sie umgesetzt wurden.

2.1 Verwandte Arbeiten

In diesem Abschnitt wird zuerst analysiert, wie Texturen entwickelt werden, um zu verstehen, welche Anforderungen an die Entwicklungsumgebung gestellt werden müssen. Dafür werden Methoden und Verfahren von aktuellen Papern vorgestellt, die sich diesem Thema gewidmet haben. Als zweiter Teil wird analysiert, welche browserbasierten Anwendungen es gibt, die zumindest einen Teil der Aufgabe einer Entwicklungsumgebung für prozedurale Texturen bereitstellen. Die Auswahl ist sehr begrenzt, da eine Schnittstelle für das Erzeugen von prozeduralen Texturen: WebGL erst seit 2009 in erster Version veröffentlicht wurde.

2.1.1 Entwicklung von Texturen

Für das Erstellen von Texturen gibt es mehrere Möglichkeiten. Einige davon werden im folgenden Teil vorgestellt und analysiert. Eine einfache Methode, Texturen zu erzeugen, ist das manuelle Zeichnen von Texturen am Computer. Das Resultat ist eine kaum fotorealistische Bitmap, die auf Objekte gemappt werden kann. Die Grenzen dieses Verfahrens zeigen sich schnell: Wenn immer feinere und komplexere Texturen benötigt werden, steigt der Aufwand für deren Erstellung enorm. Eine weitere Methode ist das Erzeugen von Photos von gewünschten Strukturen. Probleme gibt es allerdings bei Bildrändern, Beeinflussung durch Licht und Schatten sowie Artefakten, die in den Bildern auftreten können. Eine Nacharbeitung der Bilder am Computer ist deshalb in vielen Fällen notwendig. Auch hier werden Bitmaps an die Grafikkarte gesendet, was bei großen, detailreichen Bildern zu Effizienzproblemen führt. Während zum Beispiel bei neuen GPUs die Berechnungskapazitäten immer weiter steigen, bleibt die Speicherbandweite gleich. Verfahren, die Texturen nicht aus dem Speicher laden, sondern direkt in der GPU berechnen, werden somit immer effizienter. Prozedurale Texturen, die auf der CPU berechnet werden, haben auch den Vorteil, dass keine Speicherzugriffe, außer zum Laden des Textur-Codes, notwendig sind. Allerdings müssen die Textur-Daten anschließend an die GPU geschickt werden.

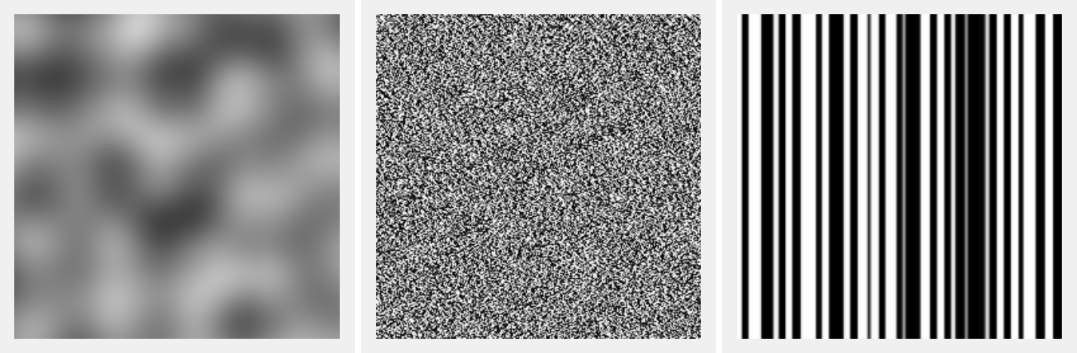

Abbildung 2.1: Mit der browserbasierten Entwicklungsumgebung erzeugte Bilder, die mit Hilfe der prozeduralen Textur „Improved Perlin Noise“ [Imp] und drei Parametern: Frequenz, Streckung und Konstrast erstellt wurden. Von links nach rechts ist die Frequenz niedrig, sehr hoch und mittel; die Streckung niedrig, niedrig, sehr hoch; der Kontrast mittel, hoch und sehr hoch.

Beispiele für Texturen die auf der CPU berechnet werden und nicht prozedural sind, werden im Paper „State of the Art in Example-based Texture Synthesis“ [WLKT09] vorgestellt. Diese Methoden basieren darauf, dass der Benutzer ein Sample eingibt und daraus eine Textur erzeugt wird. Die meisten Algorithmen basieren auf „Markov Random Fields“ (MRF) Methoden: Viele Texturen werden dabei aus einer Kombination von lokalen und stationären Prozessen berechnet. Dabei wird jeder Textur-Pixel (Texel) durch die gleiche Charakterisierung aus einer kleinen Menge von räumlichen Nachbarpixeln bestimmt. Die Charakterisierung wird durch die Eingabe eines Samples generiert. Die Pixel-basierte Synthese ist ein Beispiel für einen Basis-Algorithmus, der Example-Based Textur-Synthese, und wird hier grob vorgestellt:

1. Erstellen der Charakteristika: Zum Beispiel wird für jeden Pixel aus dem Sample eine kleine räumliche Menge von Nachbarpixeln betrachtet.
2. Erzeugen der Textur (Pixel für Pixel): Für jeden Pixel wird die Umgebung mit den Sample-Pixeln und ihren Nachbarmengen verglichen. Der Sample-Pixel mit den meisten übereinstimmenden Nachbarpixeln wird ausgewählt und auf den Pixel kopiert.

Mit der Example-Based Textur-Synthese lassen sich begrenzte Variationen von Texturen erstellen, die eine hohe Qualität und keine unnatürlichen Artefakte aufweisen. Der Nachteil ist, dass diese Methode nicht gut skaliert, da die Erzeugungskomplexität proportional zur Oberflächen- oder Textur-Größe wächst.

Eine weitere Methode, um Texturen zu erstellen, bieten prozedurale Textur-Programme. Die Textur wird durch ein im Vergleich zur fertigen Textur kleines Programm direkt auf der CPU oder sogar auf der GPU berechnet und Speicherzugriffe finden nur statt, um den Code zu laden. Die Schwierigkeit bei prozeduralen Texturen liegt im Finden passender Funktionen, um Muster zu erzeugen, die natürlich aussehen und sich nicht wiederholen. Grundlegende Verfahren für prozedurale Texturen verwenden Basis-Funktionen wie Perlin-Noise [Per85] oder Turbulence, die eine Summe von Rauschfunktionen ist. Perlin-Noise wurde von Ken Perlin entwickelt: Dabei wird zuerst eine Ebene durch ein Gitter mit

Abbildung 2.2: Mit der browserbasierten Entwicklungsumgebung erzeugte Bilder, die mit Hilfe einer prozeduralen Textur berechnet wurden, die Turbulence-Funktionen mit Sinusschwingungen kombiniert. Im zweiten und dritten Bild wurde außerdem die Textur als Heightmap verwendet, die den Farben unterschiedliche Höhen zuordnet.

quadratischen oder rechteckigen Zellen beschrieben. Die Gitterpunkte stellen ganze Zahlen dar und bekommen jeweils zufällige Gradienten zugewiesen, die reproduzierbar sind und für jede Richtung die gleiche Wahrscheinlichkeit besitzen. Dadurch, dass die Auflösung der Ebene größer ist als die des Gitters, müssen die Werte der Pixel interpoliert werden. Das geschieht, indem für jeden Punkt der Abstand zu den umliegenden Gitterpunkten mit den jeweiligen Gradienten gewichtet wird.

Das Gitter kann durch Parameter kontrolliert werden: Die Frequenz kann durch die Gittergröße, die Streckung durch die Seitenverhältnisse des Gitters und der Kontrast durch die Interpolations-Funktion geregelt werden. In Abbildung 2.1 sieht man drei Beispiele für „Improved Perlin Noise“ mit unterschiedlichen Parametern. Die Basisfunktionen reichen nicht aus, um komplexe Muster zu erzeugen. Erst in Kombination mit mathematischen Funktionen, wie zum Beispiel Sinus-Wellen, lassen sich eine größere Vielfalt von Mustern erzeugen. Im Paper [Per85] wird durch die Kombination von Turbulence-Funktionen mit einer Sinus-Schwingung die Textur „marble veins“ erzeugt. In Abbildung 2.2 ist ein Beispiel hierfür zu sehen. Um die richtige Kombination von Basistexturen und mathematischen Funktionen auszuwählen, sind gutes mathematisches Wissen und meistens weitreichende Programmierkenntnisse wichtig.

Aus diesem Grund wird im Paper „Multi-scale Assemblage for Procedural Texturing“ [GDG12] ein Verfahren vorgestellt, das ohne gute mathematische und Programmier-Kenntnisse auskommt. Dabei soll die Erzeugung von strukturierten prozeduralen Mustern durch ein Prinzip der einfachen Faltung stark vereinfacht werden. Das heißt, der Benutzer erstellt einfache Formen: Diese bestehen aus Eckpunkten und verbindenden Kanten und werden mit Hilfe von Distanzberechnungen und Interpolationen zu Figuren kombiniert. Um noch komplexere Muster zu erhalten, wird nicht die Komplexität der einzelnen Figuren erhöht, sondern diese werden miteinander hierarchisch kombiniert. Eine Vereinigung besteht somit aus einer hierarchischen Kombination von vielfältigen Figuren, Positionen und Figuren, die durch eine lineare Kombination von zufällig gewichteten Hauptmethoden entstehen. Diese Hauptmethoden werden statistisch durch eine Menge von Basis-Figuren und Primitiven extrahiert.

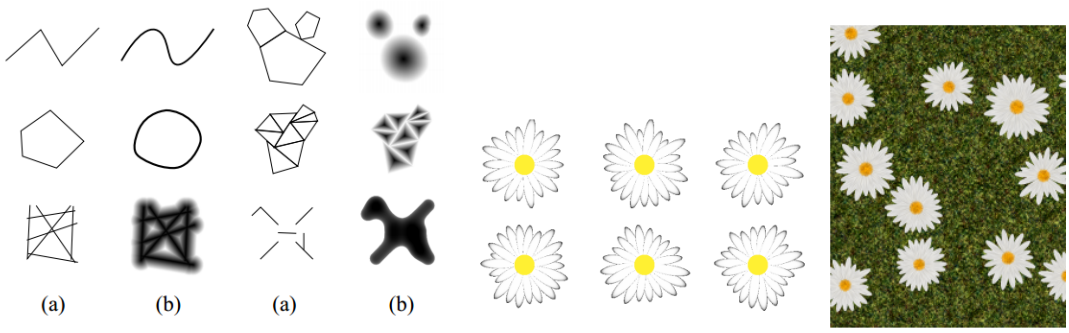

Abbildung 2.3: Beispiel für Multi-scale Assemblage for Procedural Texturing: links a) Formen die aus Polygonen bestehen, b) aus Formen erzeugte einfache Figuren, die durch Interpolation und Distanzberechnungen entstehen; Mitte, Kombination der einfachen Figuren zu komplexen Figuren; rechts, zufällige Anordnung der komplexen Figuren ergibt ein Blumenfeld (Quelle: [GDG12] Figur 3, 5, 9)

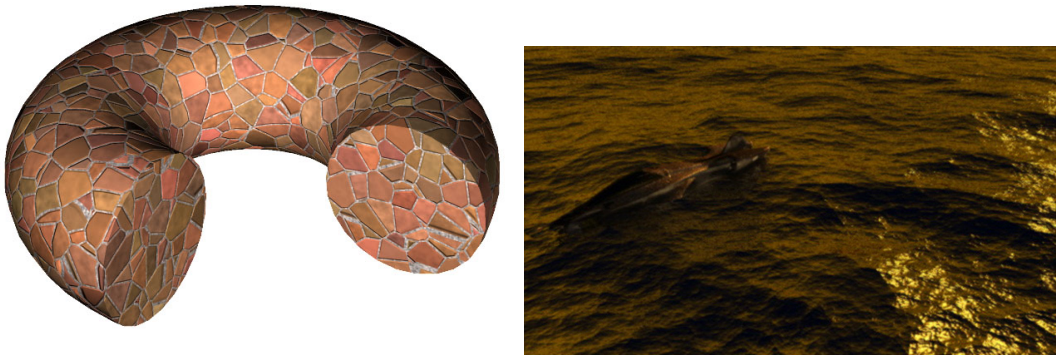

Abbildung 2.4: Mit Hilfe von „Cellular Texture“ erzeugte Bilder: links ein Teilring mit einer Pflasterstein-Struktur, rechts künstlich erzeugte Wasserwellen (Quelle: [Wor96] Anhang)

Durch dieses Verfahren erhält man eine Vielfalt von Mustern, Strukturen und komplexen Figuren, die sich auch durch Parameter kontrollieren lassen, die zum Beispiel die Kombination und Positionierung von Figuren beeinflussen.

Ein weiteres Verfahren zum Erstellen von prozeduralen Texturen wird im Paper „A Cellular Texture Basis Function“ [Wor96] beschrieben. Die Textur wird erzeugt, indem zum Beispiel in einem zweidimensionalen Raum Punkte zufällig verteilt werden. Für jeden Pixel in diesen Raum gibt es entweder einen erzeugten Punkt oder mehrere benachbarte Punkte. Durch verschiedene Abstandberechnungen, wie sie zum Beispiel euklidisch Abstandsberechnung und Manhatten Distanz, wird jedem Punkt ein Gebiet zugewiesen. In Abbildung 2.4 ist zu sehen welche Resultate bei einer passenden Farbinterpolation dieser Gebiete entstehen.

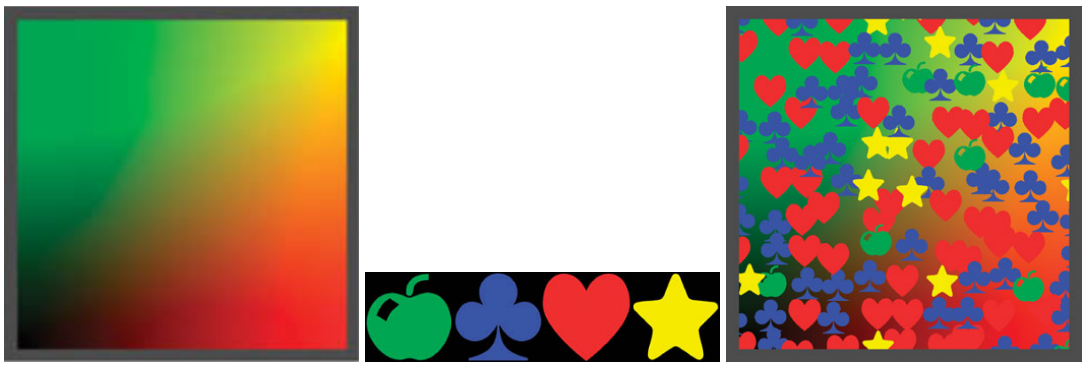

Abbildung 2.5: Beispiel für „Bombing Pattern“: links ein zweidimensionaler Raum, in der Mitte die vier verschiedenen Pattern und rechts die Kombination (Quelle: [Bom] Figure 20-1, 20-6, 20-7)

Auf der Webseite GPU Gems von Nvidia wird in Kapitel 20 [Bom] der „Bombing Pattern“ Algorithmus vorgestellt. Dieser dient dazu, zum Beispiel eine Wiese mit einer Vielzahl von Blumen zu erzeugen oder ein Stadt mit vielen ähnlichen Gebäuden, die dennoch unterschiedlich aussehen sollen. Auch hier wird der Raum in Zellen unterteilt. Für jede Zelle wird ein zufälliger Punkt berechnet und diesem wird ein Pattern zugewiesen wie es zum Beispiel in Abbildung 2.5 zu sehen ist.

Die verwandten Arbeiten zeigen, dass das Erzeugen von prozeduralen Texturen ein aufwändiger Prozess ist und in vielen Fällen der Aufwand übertroffen wird, der entsteht, wenn eine Textur zum Beispiel durch ein Photo erstellt oder am Computer gezeichnet wird. Wenn allerdings eine Prozedur erstellt ist, die eine bestimmte Aufgabe übernimmt wie zum Beispiel das Erzeugen von Holz-Mustern, und diese prozedurale Textur durch Parameter gesteuert werden kann, dann hat man nicht nur eine Textur, sondern einen Textur-Typ. Dieser Textur-Typ kann durch das passende Einstellen der Parameter auf viele Holzgegenstände mit unterschiedlichen Strukturen gemappt werden. Der Aufwand ist deshalb zuerst sehr groß, die Vorteile aber um so größer.

2.1.2 Anwendung

Ein Teilziel dieser Bachelorarbeit ist das Erstellen einer Browser-basierten Entwicklungsumgebung für prozedurale Texturen. In diesem Abschnitt werden zwei Web-Anwendungen vorgestellt und analysiert, die es ermöglichen, Fragment-Shader im Webbrowser zu programmieren und darzustellen.

Shadertoy

Shadertoy [Sha] (Abbildung 2.6) ist eine der ersten Webseiten, die es seit 2009 erlaubt, Shader mit Hilfe von WebGL darzustellen und zu programmieren. Die Webseite lässt sich in zwei Teile unterteilen: Links hat man ein WebGL-Fenster, das den Shader anzeigt und rechts einen Editor für den Textur-Shader oder einen Musik-Shader. Die zwei Editoren sind in verschiedene Tabs getrennt und unterteilen sich in Shader Eingabewerte, Hauptfunktion sowie 4 Eingabe-Channels.

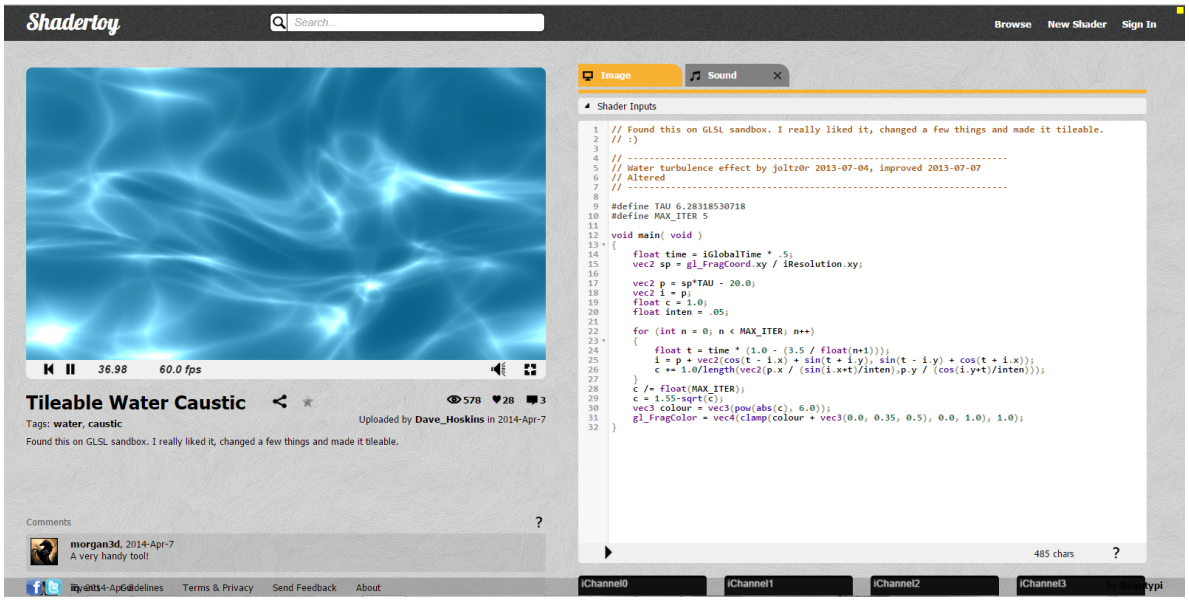

Abbildung 2.6: Screenshot der Webseite Shadertoy: links animiertes WebGL-Fenster mit Bewertungs- und Kommentarfunktion, rechts Code-Editor und Eingabe-Channels (Quelle: [Sha])

Die Eingabe-Channels erlauben das Einbinden von Tastatureingaben, Bitmap-Texturen, Videos, der WebCam, Cubemaps oder Sounddateien. Auf die Channel-Eingaben kann direkt in der Hauptfunktion durch einen Sampler zugegriffen werden. Als wichtigste statische Eingaben für die Hauptfunktion dienen die Auflösung, die aktuelle Uhrzeit, das aktuelle Datum sowie die Mausposition.

Es kann somit ziemlich einfach eine Ausgabe im WebGL-Fenster erzeugt werden, die zum Beispiel abhängig von der Zeit oder der Mausposition und anderen Eingabeparametern ist. Bis auf die vier Eingabe-Channels hat der Benutzer allerdings keine Möglichkeit, die Shader-Eingabeparameter zu bestimmen.

Der Editor bietet ein „Syntax Highlighting“, kein „on the fly“ „Syntax Checking“, intelligentes Code-Formatieren, keine Unterstützung bei der korrekten Klammerung und keine automatische Code-Vervollständigung. Falls beim Compilieren der Hauptfunktion ein Fehler auftritt, wird eine genaue Fehlermeldung angezeigt.

Aus dem Aufbau der Webseite Shadertoy ist ersichtlich, dass der Fokus im Verbreiten und Veröffentlichen von erzeugten Shadern liegt. Dafür werden die Shader online in einer Datenbank gespeichert und auf Wunsch des Benutzers veröffentlicht. Andere Benutzer können sich den Shader mit dem Code ansehen, kommentieren und bewerten. Es ist außerdem erkenntlich, wie oft ein Shader angeschaut wurde. Um einen Shader zu speichern und öffentlich zu machen, muss sich der Benutzer anmelden, notwendig dafür ist ein Benutzername eine E-Mail Adresse sowie ein Passwort. Auf der Hauptseite von Shadertoy wird der Shader der Woche sowie eine Anzahl von „featured “ und empfohlenen Shader-Resultaten angezeigt.

Shadertoy bietet eine große Anzahl von Funktionen, es treten jedoch auch Probleme auf: Zum einen werden direkt auf der Hauptseite zuerst 12 Shader ausgeführt, was auf vielen PCs zu Performance-Einbrüchen führt, zum anderen können auf kleinen Bildschirmen Webseiten-Elemente abgeschnitten werden (siehe Abbildung 2.6 Eingabe-Channels unten rechts). Der oben gezeigte Algorithmus „Improved Perlin-Noise“ braucht für die Berechnung ein großes Array, das allerdings mit der derzeitigen Shader-Sprachenversion in der Hauptfunktion nicht benutzt werden darf. Mit Shadertoy lassen sich deshalb nicht alle Shader darstellen, ein weiterer Eingabe-Parameter, durch den Arrays als Texturen importieren werden, ist deshalb notwendig. Außerdem gibt es keine Möglichkeit, auf den Vertex-Shader zuzugreifen, 3D Objekte in die Szene zu importieren oder zu erzeugen.

Wichtig bei prozeduralen Texturen ist es, Parameter zu kontrollieren und den Effekt direkt zu sehen. Ein Slider, der das ermöglichen könnte, ist nicht vorhanden. Für prozedurale Texturen ist Shadertoy nur bedingt geeignet. Die Stärken der Webseite liegen eher beim Manipulieren, Animieren, Transformieren und Kombinieren von Bitmaps und anderen Eingabeelementen sowie dem Austauschen, Bewerten und Kommentieren von Shadern.

GLSLSandbox

GLSLSandbox [GLS] (Abbildung 2.7) ist eine weitere Webseite, die allerdings im Vergleich zu Shadertoy weniger Funktionalität bietet. Auf der Startseite werden eine Reihe von Vorschaubildern von Shadern angezeigt, die der Benutzer anklicken kann. Anschließend bekommt er den Shader im Hintergrund angezeigt, während im Vordergrund ein transparenter Editor angezeigt wird. Dieser unterscheidet sich nicht wesentlich vom Editor auf der Webseite Shadertoy, er besitzt allerdings einen „on the fly“ Syntax-Checker. Für GLSLSandbox ist typisch, dass ziemlich einfach zu einem Shader ein Kind-Shader erzeugt werden kann. Der Benutzer hat dabei immer die Möglichkeit, den ursprünglichen Shader und den Kind-Shader direkt zu vergleichen.

GLSLSandbox bietet als Eingabeparameter die aktuelle Zeit, die Auflösung sowie die Mausposition. Zusätzlich dazu kann vom Benutzer bestimmt werden, in welcher Qualität der Shader angezeigt wird. Während bei Shadertoy nur die Hauptfunktion editierbar war, kann bei GLSLSandbox der komplette Fragment-Shader-Code geändert werden. GLSLSandbox dient dazu, Fragment-Shader zu programmieren, die als Parameter nur die aktuelle Zeit und die Mausposition haben. Als Entwicklungsumgebung für prozedurale Texturen ist die Webseite nicht ausreichend.

2.2 Grundlagen

In diesem Abschnitt werden zum einen die formalen Voraussetzungen von prozedurale Texturen genauer analysiert und zum anderen erklärt, welche Elemente für die browserbasierte Entwicklungsumgebung wichtig sind und welche Funktionalität sie beinhaltet.

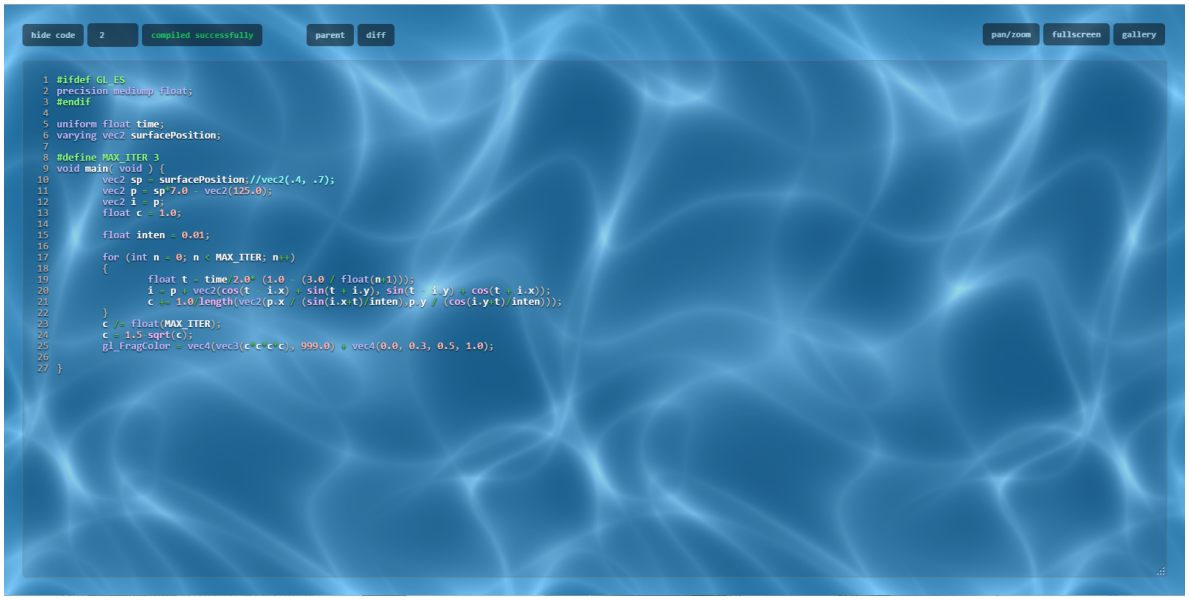

Abbildung 2.7: Screenshot der Webseite GLSLSandbox: Im Hintergrund sind ein animierter Shader sowie im oberen Bereich Kontrollelemente zu sehen, im Vordergrund wird ein transparenter Code-Editor dargestellt. (Quelle: [GLS])

2.2.1 Prozedurale Texturen

Texturen werden für die Gestaltung von Oberflächen von 3D-Modellen in der Filmbranche oder bei der Entwicklung von digitalen Spielen eingesetzt. Da der Realismus und der Detaillierungsgrad von digital erstellten Oberflächen immer weiter steigt, erhöht sich der Aufwand für die Erstellung von Texturen: Eine Textur, die zum Beispiel auf einer Bildschirmauflösung von 800 x 600 gut und realistisch ausgesehen hat, wirkt auf Bildschirmauflösungen von 1920 x 1080 oder noch höher pixelig und ungenau. Die Textur muss deshalb größer werden was für manuell gezeichnete Texturen einen höherer Aufwand bedeutet. Bei Photo-Texturen steigt der Aufwand beim digitalen Nachbearbeiten, da jedes noch so kleine Artefakt, das beim Wrapping entsteht, bei höheren Bildschirmauflösungen sichtbar wird. Außerdem wird bei beiden Verfahren nur eine begrenzte Datenmenge zur Verfügung gestellt.

Prozedurale Texturen ermöglichen es, diesen Aufwand zu verringern. Dabei wird eine Textur nicht gezeichnet oder durch ein Photo erzeugt, sondern der Designer erstellt eine Berechnungsvorschrift, durch welche die Textur automatisch erzeugt werden kann. Auch wenn das Erstellen der Berechnungsvorschrift in vielen Fällen genauso aufwändig oder sogar noch aufwändiger ist als die Erstellung der Textur durch manuelles Zeichnen oder durch ein Photo, bietet eine prozedurale Textur die Möglichkeit, einfache Variationen zu erstellen. Dadurch wird nicht nur eine Textur erzeugt, sondern es können viele Texturen vom selben Typ generiert werden.

Es gibt zwei wesentliche Modelle, um Texturen zu berechnen: Entweder kontextsensitiv, das bedeutet, jeder Pixel kennt den Zustand der ihn umgebenden Pixel oder kontextunabhängig. Im Allgemeinen

gelten nur kontextunabhängige Verfahren als prozedural. Die in dieser Arbeit und von der Webseite erzeugten Texturen sind kontextunabhängig und somit auch prozedural. Das erlaubt einen sehr hohen Grad an Parallelität und ist für die schnelle Berechnung in der Grafikkarte wichtig.

Ein weiterer Vorteil von prozeduralen Texturen ist der geringe Speicherbedarf. Anstatt Texturen in Form von Bildern zu speichern und zum Beispiel bei digitalen Spielen durch permanent hohen Speicherzugriff zu laden, können kleine Berechnungsvorschriften in die Grafikkarte geladen werden, und der schnelle Grafikprozessor übernimmt die Berechnung von Texturen. Da der Flaschenhals von CPU- und GPU-Berechnungen meist der Speicherzugriff ist, kann dadurch die Effizienz und Leistung beim Erzeugen von Filmen oder bei der Ausführung von digitalen Spielen gesteigert werden.

Indem Kontrollparameter für eine prozedurale Textur erstellt werden, können bestimmte Eigenschaften wie zum Beispiel die Farbe, der Kontrast oder die Frequenz der Muster kontrolliert werden. Das hilft dem Designer, eine prozedurale Textur nach bestimmten Anforderungen zu erstellen.

Die Ersetzung einer Standard-Bild-Textur durch eine prozedurale Textur wird durch Kontrollparameter vereinfacht. Gibt es eine Metrik, welche die Ähnlichkeit beider Ergebnisse bewertet, so kann die Anpassung der Parameter auch automatisiert oder zumindest zum größten Teil automatisiert stattfinden. Ein Beispiel hierfür wird im Paper „Interactive Parameter Retrieval for Two-Tone Procedural Textures“ [GKHF14] vorgestellt.

2.2.2 WebGL

Um prozedurale Texturen im Browser plattformunabhängig zu berechnen und darzustellen, braucht es eine 3D-Graphik-Programmierschnittstelle. Das ist sinnvoll, da die GPU prozedurale Texturen schneller berechnen kann und einen höheren Grad an Parallelität erlaubt. Als wichtigster Standard wird WebGL [KHR] seit 2009 von der Khronos Group und Mozilla lizenzfrei entwickelt und wurde 2011 in erster Version veröffentlicht. WebGL basiert auf OpenGL ES 2.0, eine Sprachen-unabhängige vereinfachte Version von OpenGL. Allerdings ist die derzeitige Version von WebGL eher vergleichbar mit der der OpenGL Version 2.0, welche im Gegensatz zur aktuellen OpenGL Version 4.5 viele Funktionen noch nicht unterstützt.

Dadurch entstehen Probleme beim Implementieren von prozeduralen Texturen, die auf bestimmte Operationen angewiesen sind. Ein Beispiel hierfür ist die „Improved Perlin Noise“ Textur [Imp], die Bit-Shifting Operationen zur Berechnung benötigt und auf ein großes Array zugreift. Beides wird aktuell nicht unterstützt, wobei das Array durch eine Textur ersetzt werden kann und Bitshifting mittels dem Modulo Operator simuliert werden kann. Diese Lösung ist sehr ineffizient.

Das Programmieren der prozeduralen Texturen für die GPU funktioniert über Shader. Die wichtigsten zwei Shader-Typen sind der Vertex-Shader und der Fragment-Shader. Der Vertex-Shader erlaubt es, die Geometrie eines 3D-Objektes zu manipulieren (Abbildung 2.8), und der Fragment-Shader ist für die Berechnung und das Mappen der Texturen auf die Oberflächen zuständig. Ein weiterer Vorteil ist, dass Shader als Text-Strings an die Programmierschnittstelle gesendet werden und deshalb nicht vorcompiliert werden müssen.

Für die browserbasierte Entwicklungsumgebung wird das Framework Three.js [THR] verwendet, das eine umfassende und freie, in Javascript implementierte 3D-Umgebung bietet. Somit werden Aufgaben

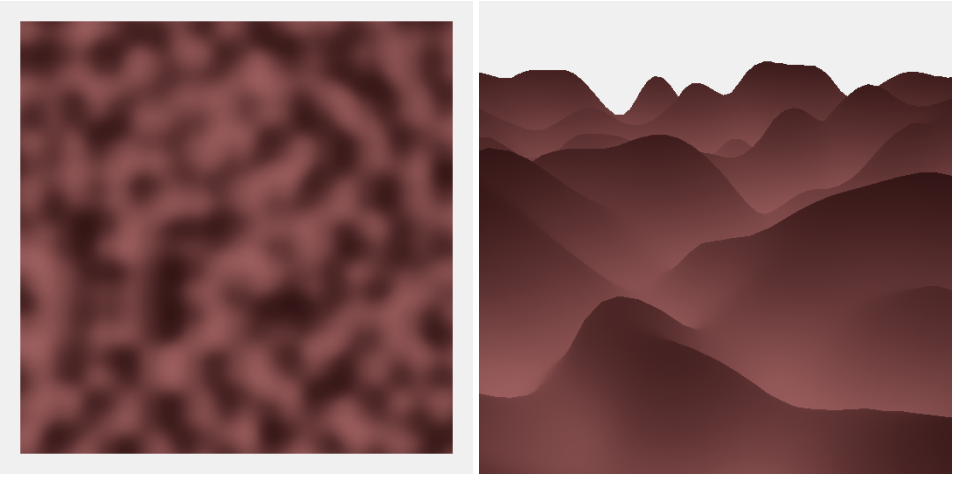

Abbildung 2.8: Mit der browserbasierten Entwicklungsumgebung erstellte Bilder: links ist eine Perlin-Contrast-Textur mit einem Standard Vertex-Shader, der das Objekt nicht verändert, zu sehen und rechts ist die selbe Textur mit einem anderem Vertex-Shader zu sehen. Dieser Vertex-Shader skaliert, rotiert und transformiert die Ecken des Objekts.

Webbrowser	unterstützt ab Version
Internet Explorer	11
Mozilla Firefox	5
Google Chrome	12
Safari	5.1
Opera	12
Google Chrome für Android	37

Tabelle 2.1: Webbrowserunterstützung für WebGL

wie das Erstellen von 3D-Objekten und das Hinzufügen von Texturen und Shadern stark vereinfacht. Es ist zusätzlich möglich, mit einem externen 3D-Programm wie zum Beispiel Blender [Ble14] oder Maya [may] eine 3D-Szene zu erstellen und diese anschließend mit Three.js zu importieren.

Tabelle 2.1 beschreibt, ab welcher Version verschiedene Webbrowser WebGL unterstützen.

2.2.3 Editor

Der Code prozeduraler Texturen wird in einer Entwicklungsumgebung geschrieben, die den Benutzer unterstützt. Essentiell dabei ist das Textfenster. Da sich Textur Code an syntaktische und semantische Regeln halten muss, eignen sich HTML-Source-Code-Editoren sehr gut, die ein solches Highlighting

umsetzen. Es gibt eine große Auswahl an Open-Source-Editoren. Um einen passenden auszuwählen, wird hier ein Vergleich der unterstützten Features durchgeführt. Dabei sind für die Webseite eine schnelle, einfache und übersichtliche Bedienung des Editors wichtig, um auch ungeschulten Benutzern ohne lange Einarbeitungszeit das Programmieren von prozeduralen Texturen zu ermöglichen. Die Editoren werden auf Features wie „Syntax Highlighting“ (SH), „Syntax Checking“ (SC), Intelligentes Code Formatieren (ICF), Intelligentes Klammern (IK), Automatische Code Vervollständigung (ACV), Zeilen-Nummerierung (ZN) und UNDO/REDO Speicherung (U/R) überprüft. Im folgenden sind die Anforderungen an die Features aufgelistet:

SH Der Editor erkennt nach der Einstellung der Programmiersprache automatisch die Syntax und färbt den eingegebenen Code automatisch ein. Falls ein Editor nur bestimmte Programmiersprachen beherrscht, sollte zumindest die Möglichkeit bestehen, weitere Syntax-Definitionen zu erstellen.

SC Der Editor beherrscht einfache Mechanismen, um grobe Syntaxfehler zu erkennen und zu markieren.

ICF Der Editor hilft dem Benutzer bei der Code-Eingabe durch automatisches Einrücken und Erkennen von Programm-Blöcken. Hierbei wird nicht verlangt, dass der Editor per Klick eine Formatierung durchführt, sondern dass er das während des Schreibens unterstützt.

IK Der Editor erkennt Programmblöcke und hilft bei der richtigen Klammerung.

ACV Der Editor bietet eine automatische Vervollständigung von Schlüsselwörtern an.

ZN Der Editor ermöglicht die Verwendung von Zeilennummern.

U/R Der Editor speichert Text-Eingabe-Zustände und ermöglicht dem Benutzer, mit Einschränkungen zwischen diesen Zuständen zu wechseln.

Zusätzlich zu den Editoren ist in der Tabelle für den Vergleich eine einfache HTML „textarea“ aufgeführt.

Ace-Editor

Für die browserbasierte Entwicklungsumgebung für prozedurale Texturen wurde der Ace-Editor ausgewählt: Er ist extrem effizient, unterstützt über 110 verschiedene Programmiersprachen und eine große Auswahl an verschieden Editor-Styles. Auch beim Ace-Editor ist die Syntax-Überprüfung für Javascript und Shader-Sprache nicht ausreichen, um fehlerfreien Textur-Code zu erzeugen. Für Javascript-Code war die Implementierung eines Parsers notwendig und für die Shader-Sprache eine Vorcompilierung, um ausreichend Feedback zu erzeugen. Durch die Erweiterung des Ace-Editors erfüllt er alle Anforderungen, die an einen passenden Editor gestellt werden.

Editor	Features						
	SH	SC	ICF	IK	ACV	ZN	U/R
Ace Editor	ja	ja	ja	ja	(ja)	ja	ja
Code Mirror	ja	(ja)	ja	(ja)	ja	ja	ja
Markitup	nein	nein	nein	nein	nein	(nein)	ja
Ymacs	(ja)	nein	(ja)	(ja)	(ja)	ja	ja
Code Press	ja	nein	(nein)	nein	(ja)	ja	ja
Edit Area	(ja)	nein	(nein)	nein	nein	ja	ja
TextArea	nein	nein	nein	nein	nein	(nein)	(ja)

Tabelle 2.2: Vergleich von HTML-Code-Editoren anhand von Features wie „Syntax Highlighting“ (SH), „Syntax Checking“ (SC), Intelligentes Code Formatieren (ICF), Intelligentes Klammern (IK), Automatische Code Vervollständigung (ACV), Zeilen Nummerierung (ZN) und UNDO/REDO Speicherung (U/R). Als Bewertungen gibt es „ja“, „nein“, „(ja)“. „(JA)“ bedeutet, dass der Editor ein Feature zum großen Teil und „(nein)“, dass der Editor ein Feature kaum anbietet.

2.2.4 Datenbank

Um Texturen-Code dynamisch und komfortable zu speichern, zu laden und auszuführen, braucht es eine Datenbank. Im Gegensatz zu Programmen, die nicht auf einen Webbrowser angewiesen sind und somit auf lokale Dateien dynamisch zugreifen können, wird bei Webseiten aus Sicherheitsgründen ein dynamischer Dateizugriff verhindert.

Das Ziel ist es, eine Datenbank für HTML zu finden, die ohne zusätzliche Programme oder Server auskommt und dennoch zuverlässig und schnell funktioniert. Eine MySql Datenbank, repräsentativ für Serverseitige Datenbanken, ist somit nicht passend. Für Clientseitige Datenbanken hat sich Indexeddb [ind] als Standard etabliert und wird von fast allen aktuellen Webbrowsern unterstützt. Die Speicherung der Daten wird komplett vom Browser übernommen, ohne dass zusätzliche Scripte verwendet werden müssen. Indexeddb ist eine asynchrone Datenbank, die geeignet ist, große strukturierte Datenmengen zu speichern. Die Asynchronität der Datenbank führt dazu, dass alle Teile, die von der Datenbank abhängen, Event-basiert programmiert werden müssen. Für die Kommunikation und Speicherung der Daten wird keine SQL-Syntax benutzt, sondern Objekte werden direkt in einem Objekt-Speicher abgelegt. Da Indexeddb eine lokale Datenbank ist, kann damit kein Textur-Code auf dem Server gespeichert werden. Für einen komfortablen Austausch zwischen den Benutzern ist eine weitere Datenbank notwendig. <Bild zur Verdeutlichung/ Funktionsweise>

2.3 Implementierung

Dieser Abschnitt beschreibt die Umsetzung der browserbasierte Entwicklungsumgebung, welche Anforderungen erfüllt wurden und wie die fertige Entwicklungsumgebung funktioniert. Zur Verdeutlichung werden einige Beispiele gezeigt.

2.3.1 Technische Anforderungen

An die Implementierung der Browser-basierten Entwicklungsumgebung werden Anforderungen gestellt, welche im Folgenden aufgelistet sind: Als Hauptanforderung sollte prozeduraler Textur-Code editierbar dargestellt werden. Dafür kann ein Source-Code-Editor verwendet werden, der den Benutzer bei der Eingabe unterstützt und die Einarbeitung stark erleichtert. Das heißt, dass der Editor als wichtigstes Feature einen „Syntax Checker“ braucht, der die Eingabe so weit überprüft und Feedback bei falscher Eingabe gibt, so dass der Code fehlerfrei compiliert werden kann.

Da Shader die große Rechenkapazität der GPU benutzen können, sollte ein Shader-Editor implementiert sein. Um eine Vielzahl von Texturen zu unterstützen, ist es nicht ausreichend, nur einen Editor zum Shader-Programmieren zu implementieren, da sonst Basis-Texturen wie „Perlin Contrast“ nur schwer realisiert werden können. Deshalb sollten zusätzlich Texturen mit einer Hochsprache wie zum Beispiel Javascript erzeugt werden können.

Als weitere Anforderungen sollte der Textur-Code mit Hilfe von WebGL graphisch gerendert und angezeigt werden können. Das Programm muss außerdem Parameter der Textur automatisiert erkennen und passende Slider zur Kontrolle dieser Parameter anbieten.

Um die Benutzererfahrung mit der Entwicklungsumgebung zu verbessern, sollte ein System zum Speichern und Laden von Texturen vorhanden sein. Die Implementierung einer Datenbank bietet sich dafür an.

Als zweiter Teil sollte die Implementierung das Laden und Anzeigen eines Referenzbildes ermöglichen, sowie Interaktionen mit diesem, wie zum Beispiel das Extrahieren einer Farbe mit Hilfe eines Color-Pickers.

Um Rückschlüsse auf eine automatisierte Einstellung der Parameter zu bekommen, um ein spezielles Design umzusetzen, muss eine Technik zum Aufzeichnen der Benutzerinteraktion implementiert sein. Dafür ist es ausreichend, die Bewegung und Eingabe der Maus sowie die Eingabe der Tastatur zu speichern. Für detailliertere Aufzeichnungsmethoden wie zum Beispiel „Eye Tracking“ wird externe Software benötigt.

2.3.2 Webseite

Wie man in Abbildung 2.9 sehen kann, ist die Webseite grob in zwei Hälften unterteilt. In der linken Hälfte sieht man ein Fenster, in dem die prozedurale Textur gerendert dargestellt wird und ein Fenster für ein Referenzbild. Die rechte Seite zeigt eine Tab-Struktur an. Der erste Tab liefert ein Array von Editoren, um Texturen in der Hochsprache Javascript zu erzeugen. Der zweite Tab bietet ein Array

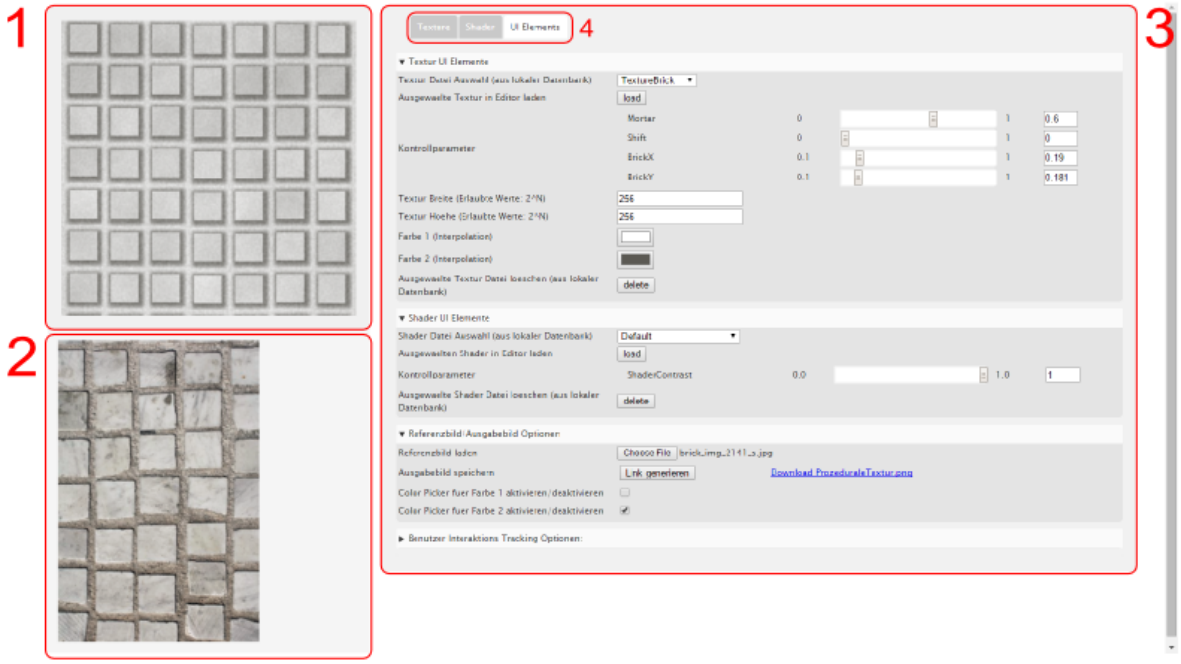

Abbildung 2.9: Grobe Aufteilung der Browserbasierten Entwicklungsumgebung: 1: WebGL-Fenster, 2: Referenzbild, 3: Javascript-Textur-Editor, Shader-Editor oder UI-Elemente, 4: Tab-Auswahl

von Editoren für die Shader Programmierung. Im dritten und letzten Tab werden dynamisch und statisch generierte Ui Elemente angezeigt.

Javascript Textur Editor

Das Textur-Editor-Array (siehe Abbildung 2.10 bietet drei Kontrollelemente: Laden einer Textur, Downloaden einer Textur und Speichern einer Textur in die Datenbank.

Um die Bedienbarkeit des Editors zu verbessern und den Aufwand für das Programmieren von Javascript-Textur-Code zu verringern, ist der Editor in ein Array von Editoren unterteilt. Für die ersten Editoren wurden die meisten Funktionen wie zum Beispiel Zeilennummerierung, „Syntax Highlighting“, erweitertes „Syntax Checking“ komplett abgeschaltet, da die Funktionen für die Eingabe von Dateiname, Dateityp, Namespace nicht notwendig sind. Des Weiteren gibt es einen Editor, um eine Konstante zu erzeugen, die bei den meisten anderen Editoren ohne zusätzliche Referenz verwendet werden kann. Durch ein Plus-Symbol lassen sich beliebig viele Editoren für Konstanten erzeugen.

Für die Hauptfunktion, die schließlich zum Erzeugen der Textur aufgerufen wird, gibt es einen weiteren Editor. Durch das Klicken einer „Checkbox“ wird ein erweitertes „Syntax Checking“ aktiviert, das allerdings eine striktere Syntax als viele Javascript-Compiler erfordert. So muss jede Variable ordentlich deklariert sein, unabhängig davon, ob eine Funktion, ein Objekt, ein Array oder ein Wert

Abbildung 2.10: Javascript-Textur-Editor der browserbasierten Entwicklungsumgebung: Oben sind drei Kontrollelemente: Laden, Downloaden, Speichern in Datenbank, zu sehen und darunter die Unter-Editoren für den Dateinamen, Dateityp, Namespace, die Konstanten (beliebig erweiterbar: „Plus Button“), die Parameter, die Haupt-Berechnungsfunktion sowie für die Hilfs-Funktionen.

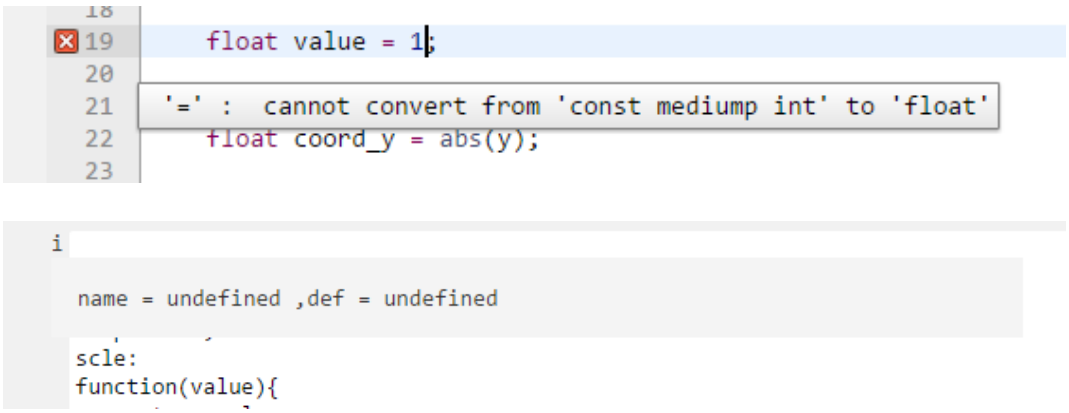

Abbildung 2.11: Das obere Bild zeigt, wie der Shader-Editor auf einen Eingabe-Fehler bei eingeschaltetem erweiterten „Syntax Checker“ reagiert. Analog funktioniert das auch für einen Teil der Javascript-Editoren. Im unteren Bild ist die Definition eines Parameters angezeigt, bei der Name und Default-Wert fehlen: Beim Verlassen des Editors wird ein Informations-Symbol mit entsprechender Fehlermeldung angezeigt.

zugewiesen oder gelesen wird. Das hilft, Fehler zur Kompilierzeit zu vermeiden, die zum Beispiel durch Tippfehler entstehen.

„Math“ ist das einzige vordefinierte Objekt. Es bietet wichtige mathematische Funktionen und Konstanten, die zur Berechnung von prozeduralen Funktionen notwendig sind: Math.sin(x), Math.cos(x), Math.pow(x,y), Math.E, Math.PI. Eine vollständige Referenz für das Math-Objekt findet sich in: [wsc]. Der Benutzer muss allerdings bei der Benutzung von Objekten vorsichtig sein: Zugriffe auf Objekt-Eigenschaften oder -Methoden, die nicht existieren, liefern keinen Fehler. Beim Rendern der kompilierten Textur kann somit zum Beispiel ein schwarzes Bild erscheinen.

Die Komplexität der Hauptfunktion wird stark reduziert, indem weitere Editoren, die für Hilfsfunktionen zuständig sind, eingefügt werden. Diese lassen sich beliebig oft duplizieren und bieten dieselbe Funktionalität wie der Editor für die Hauptfunktion.

Der nächste Typ von Editor ermöglicht dem Benutzer, Parameter für die prozedurale Textur zu erzeugen. Für jeden Parameter muss ein Name, ein Minimum- und Maximum-Wert, eine Schrittweite, ein Standardwert, sowie eine Funktion zum Skalieren des Parameters angegeben werden. Falls bei der Definition eines Parameters ein Wert vergessen wurde, wird der Editor beim Verlassen mit einem Informationssymbol versehen. Durch ein „Maus-Over“ bekommt der Benutzer genaue Informationen über fehlende Werte.

Der erweitere „Syntax Checker“ lässt sich optional für den Hauptfunktons-Editor, der die Funktion enthält die zum Berechnen der prozeduralen Textur aufgerufen wird, sowie für die Hilfsfunktionen aktivieren. Der vom Editor bereitgestellte „Syntax Checker“ wird durch einen dafür entwickelten Parser erweitert. Dieser Parser ist eine Abwandlung des Algorithmus [DCr] von Douglas Crockford, der die „Top Down Operator Precedence“ Parser Technik, beschrieben im Paper [Pra73] umgesetzt hat. Diese Technik basiert darauf, dass der Programm-Code in Tokens unterteilt wird. Diese Tokens werden von vorne nach hinten rekursiv durchgearbeitet. Wenn der „Syntax Checker“ einen Fehler gefunden hat, wird dieser mit einer detaillierten Fehlermeldung in der passenden Zeile angezeigt.

Um den Textur-Code zu speichern oder zu laden, wird der Inhalt der Editoren in das JSON-Format umgewandelt. Bevor die prozedurale Textur berechnet werden kann, wird der JSON-String vorcompiliert und in ein Script umgeschrieben. Dies läuft als Hintergrundprozess und funktioniert problemlos, falls der erweiterte „Syntax Checker“ keine Fehler anzeigt.

Im UI-Elemente Tab (siehe Abbildung 2.12) gibt es weitere Interaktions-Möglichkeiten mit dem Javascript-Textur-Editor: Zum einen kann der Benutzer hier die prozedurale Textur aus der lokalen Datenbank auswählen, die gerendert wird. Diese lässt sich per Button auch direkt in den Editor laden oder komplett aus der Datenbank löschen. Außerdem kann die Größe der Textur in 2^N Schritten (zum Beispiel: 64 x 64, 128 x 128, 512 x 64, 1024 x 2) ausgewählt werden. Die Textur-Größe beliebig groß zu wählen, ist nicht sinnvoll, da das WebGL-Fenster eine Skalierung auf 512 x 512 vornimmt.

Im Javascript Textur Editor wird für jeden Pixel ein Wert berechnet. Die Farbe des Pixel wird durch eine Interpolation von zwei Farben berechnet, die im UI Elemente Tab angegeben werden. Für die Parameter, die der Benutzer für eine Textur definiert hat, werden dynamisch Slider erzeugt. Eine Veränderung der Slider führt zum automatischen Neu-Berechnen und Anzeigen der Textur.

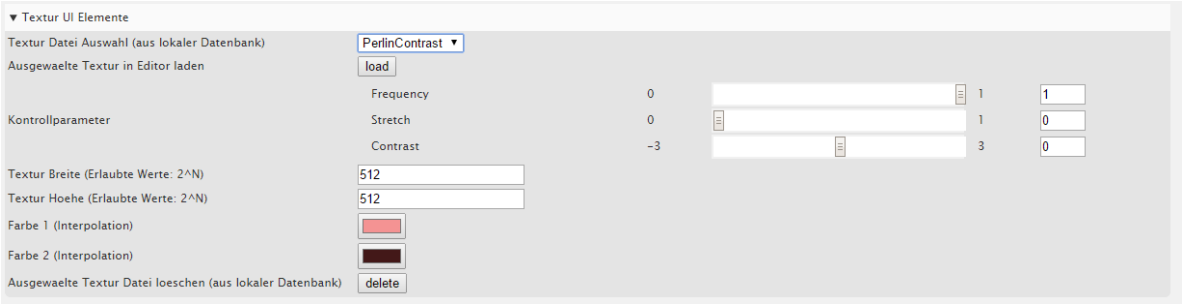

Abbildung 2.12: UI-Elemente für die Javascript-Textur aus der browserbasierten Entwicklungsumgebung: Auswahl der Textur-Datei, Laden der Textur-Datei in den Editor, Kontrollparameter, Auflösung der Textur, Farbauswahl für Interpolation, Löschen der ausgewählten Textur-Datei aus der Datenbank.

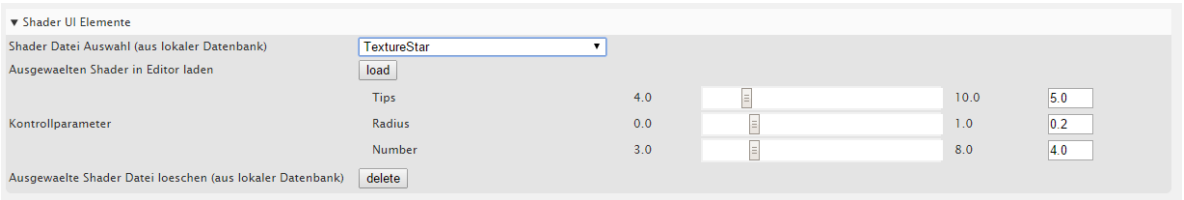

Abbildung 2.13: UI-Elemente für die Shader-Textur aus der browserbasierten Entwicklungsumgebung: Auswahl der Textur-Datei, Laden der Textur-Datei in den Editor, Kontrollparameter, Löschen der ausgewählten Textur-Datei aus der Datenbank.

Shader-Editor

Die Struktur des Shader-Editor (siehe Abbildung 2.13) unterscheidet sich in einigen Punkten vom Javascript-Editor. So gibt es keinen eigenen Editor für Konstanten oder Hilfsfunktionen oder für den Namespace. Der Editor für die Hauptfunktion unterteilt sich hier in einen Editor für den Vertex-Shader und in einen Editor für den Fragment-Shader. Im Parameter-Editor bleibt alles gleich, außer dass es keine Funktion zum Skalieren der Parameter gibt.

Die beiden Shader müssen komplett bis auf die vom Three.js Framework bereitgestellten „Uniforms“ und Attribute angegeben werden. Mit Three.js wird ein Plane erstellt, das 100 x 100 Ecken hat. Der Vertex-Shader wird für jede Ecke einmal aufgerufen und führt die in der Main Methode angegebene Berechnung aus. Der Fragment-Shader berechnet für jeden Pixel die Farbe. Standardmäßig ist ein Default-Shader voreingestellt, der die mit Javascript berechnete Textur einliest und direkt weiterleitet.

Um auf mathematische Operationen zuzugreifen, muss hier nicht wie beim Javascript-Textur-Editor das „Math“ Objekt benutzt werden, da diese Operationen direkt für die Shader-Sprache verfügbar sind. Genauere Informationen zu den unterstützten mathematische Operationen sind in der „WebGL Reference Card“ [Web] zu finden.

Auch für den Vertex- und Shader-Editor gibt es einen optional auswählbaren, erweiterten „Syntax Checker“. Dabei wird der Shader-Code probeweise vorcompiliert und die Kompilier-Fehler werden detailliert an den entsprechenden Zeilen angegeben (siehe Abbildung 2.11). Auch hier muss der Benutzer aufpassen: Es wird nicht überprüft, ob die im Parameter Editor erstellten „Uniforms“ zu den im Shader aufgerufenen passen.

Im UI-Elemente-Tab gibt es für den Javascript-Texture-Editor mit ein paar Unterschieden dieselben Optionen für den Shader-Editor. Weder lässt sich die Auflösung einstellen noch kann man Farb-Parameter an den Shader übergeben. Falls Farbwerte zum Interpolieren benötigt werden, können diese mit ein bisschen Aufwand über den Javascript-Textur-Editor bereitgestellt werden. Zum Beispiel könnten 6 Pixel der Javascript-Textur beide Farbwerte speichern und vom Shader ausgelesen werden.

Zum Ausführen der Webseite wird Chrome empfohlen. Firefox hat unter anderem Beispiel Probleme mit den Worker-Clients des Ace Editors. Diese sind für das Überprüfen der Syntax notwendig. Außerdem werden HTML5 Elemente wie „<details>“ und „<summary>“ nicht unterstützt, die für das Layout der browserbasierten Entwicklungsumgebung benutzt werden. Mit anderen Browsern wurde die Webseite nicht vollständig getestet, um eine Aussage über die Kompatibilität zu treffen.

Die Implementierung des Referenzbildes wird in Kapitel 3 behandelt und die Implementierung des Benutzer-Interaktionstracking wird in Kapitel 4 genauer beschrieben.

2.3.3 Textur-Shader-Beispiel

Die Prozessschritte, die zur Erzeugung einer prozeduralen Textur mit der browserbasierten Entwicklungsumgebung notwendig sind, werden anhand eines Anwendungsbeispiels gezeigt.

Erstellung einer prozeduralen Javascript-Textur

In den grau unterlegten Feldern sieht man gekürzte Eingabebeispiele.

1. Schritt: Auswählen eines Dateinamens. Es sind alle Kombinationen von Zahlen, Buchstaben, sowie Binde- und Unterstriche erlaubt. Nachdem die Textur in die Datenbank geladen wurde, kann die Textur im UI-Elemente-Tab anhand des Dateinamens ausgewählt werden. Die Dateinamen müssen eindeutig sein. Falls eine Javascript-Textur denselben Dateinamen wie eine schon vorhandene Textur hat, wird diese überschrieben!
   ```
   PerlinContrast
   ```
2. Schritt: Auswählen des Textur Types: Für prozedurale Javascript-Texturen ist nur: „texture“ gültig.
   ```
   texture
   ```
3. Schritt: Auswählen des Namespaces, der für die Javacript-Datei notwendig ist. Es sind nur Großbuchstaben erlaubt.
   ```
   PERCON
   ```

4. Schritt: Definieren von globalen Konstanten: Für jede Konstante muss der Name mit einem Doppelpunkt getrennt vom Wert angegeben werden. Gültige Werte sind Integer, Floats, Arrays, String und Objekte. Außerdem ist pro Editor nur eine Konstante erlaubt.

```
values : [ 151, 160, 137, 91, 90, 15, ... ]
```

5. Schritt: Definieren der Parameter, die zur Manipulation der fertigen Textur verwendet werden: Für jeden Parameter müssen mehrere Attribute, die durch Komma getrennt angegeben werden, definiert sein. Die Attribute sind Name, Minimum und Maximum Wert, Schrittweite, Default-Wert und eine Funktion, die den Parameter skaliert. Für jeden neuen Parameter muss über das Plus-Symbol am linken Rand ein neues Eingabefeld erzeugt werden.

```
name : frequency,
min : 0.0,
max : 1.0,
step : 0.01,
def : 1.0,
scale :
function(value){
        return Math.pow(2.0, 2.0 + value * 6.0);
}
```

6. Schritt: Eingeben der Hauptfunktion, die aus den Konstanten und Parametern und der aktuellen Pixelposition einen Float-Wert berechnet. Deshalb muss die Funktion einen Return-Wert haben. In diesem Editor können Konstanten, Parameter und Hilfsfunktions-Namen direkt verwendet werden. Neben dem Editor Name „Evaluate“ befindet sich eine „Checkbox“. Falls diese aktiviert ist, wird der erweiterte „Syntax Checker“ verwendet und der Benutzer bekommt Fehler „on the fly“ angezeigt. Grundsätzlich gilt die Javascript-Syntax zum Ausformulieren der Funktion. Für mathematische Operationen wird das im Abschnitt 2.3.2 beschriebene „Math“ Objekt angeboten.

```
function evalute(x,y,z){
        var t = noiseNormalized((1.0 - stretch) * x *frequency, y * frequency, 0.2);
        var value = 0.0;
        var alpha = Math.exp(contrast);
        value = (t > 0.5) ? (1.0 - (Math.pow((1.0 - t) * 2.0, alpha) * 0.5)) : (Math.pow((t
            * 2.0), alpha) * 0.5);

        return value;
}
```

7. Schritt: Zur Vereinfachungen der Hauptfunktion können im Hilfs-Funktionen-Editor zusätzliche Funktionen definiert werden. Hier gelten dieselben Regeln wie im Hauptfunktions Editor.

```
function noiseNormalized(x,y,z){
        return (noise(x,y,z) + 1.0) / 2.0;
}
```

8. Schritt: Überprüfung, ob alle Eingaben richtig sind und kein Editor eine Warnung ausgibt. Falls das zutrifft, kann der Benutzer die Textur in der lokalen Datenbank abspeichern oder einen Link erzeugen und die Textur herunterladen. Sobald die Textur in der Datenbank gespeichert ist, wird sie im UI-Elemente-Tab (siehe Abbildung 2.12) in der Option „Textur Datei Auswahl“ angezeigt und kann ausgewählt werden. Der Default-Shader ist so programmiert, dass er die Javascript-Textur ausliest und auf das Standard-Plane-Objekt mappt.

Erstellung einer prozeduralen Shader-Textur

1. Schritt: Zuerst muss der Shader-Tab ausgewählt werden.
2. Schritt: Eingeben des Dateinamens und des Dateityps. Für den Dateinamen gelten dieselben Restriktionen wie für den Dateinamen der Javascript-Textur. Als Dateityp muss „shader“ angegeben werden.
3. Schritt: Definition der Parameter. Auch hier gelten dieselben Regeln wie für Javascript-Texturen, mit einer Ausnahme. Es muss keine Skalierungs-Funktion angegeben werden. Die Skalierung wird, falls gefordert, im Fragment oder Vertex-Shader angegeben.

```
name: radius ,
min :  0.0,
max :  1.0,
step  :  0.1,
def  :  0.2,
```

4. Schritt: Erstellen des Vertex-Shaders: Im Fragment-Shader wird für jeden Pixel anhand seiner Position ein Wert oder eine Farbe berechnet. Das Attribut „myPosition“ darf allerdings wie alle anderen Attribute auch nur im Vertex-Shader gelesen werden. Damit es im Fragment-Shader verfügbar ist, muss eine Weiterleitung mittels „varying“ Variable vorgenommen werden. Deshalb wird zuerst das Attribut „myPosition“ deklariert, das die Positionen der Ecken auf einen 3D Würfel mit Länge 1 mappt.

   Als Objekt steht dem Vertex-Shader ein Plane-Objekt zur Verfügung, das in Quadrate unterteilt ist und somit aus mehr als 4 Ecken besteht. Damit das Plane-Objekt richtig dargestellt wird, müssen die Vertex-Koordinaten in View-Port-Koordinaten umgewandelt und der Variable „gl_Position“ zugewiesen werden. Dafür sind vom Framework „Three.js“die „projectionMatrix“, die „modelViewMatrix“ und der 3D-Vektor „position“ bereitgestellt. Mit weiteren Matrix-Operationen kann das Plane skaliert, rotiert und transformiert werden. Um zum Beispiel eine Highmap zu erzeugen, muss dem Z-Wert des „position“ Vektors ein Offset hinzugefügt werden.

   Falls zur Berechnung der Vertex-Positionen Parameter definiert wurden, können diese als Uniforms vom Typ Float deklariert werden. Die im Textur-Tab programmierte Javascript-Textur kann sowohl im Vertex als auch im Fragment-Shader als uniform vom Typ „sampler2D“ mit dem Namen „texture“ verwendet werden. Zugriff auf die Textur-Daten kann durch die Funktion „sampler2D(Textur Name, Position)“ erlangt werden. Dabei sind für die Position 2D-Vektoren im Bereich $\begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix}$ bis $\begin{pmatrix} 1.0 \\ 1.0 \end{pmatrix}$ relevant.

   Um den Shader probeweise zu compilieren und auf Fehler zu testen, kann am linken Rand eine „Checkbox“ aktiviert werden. Falls Fehler im Code vorhanden sind, werden die betreffenden Zeilen markiert und mit einer Fehlermeldung versehen.

```
attribute  vec3 myPosition;
varying  vec3  vPosition ;

void  main()  {
     vPosition  = myPosition;
     gl_Position  = projectionMatrix  *  modelViewMatrix * vec4(position ,1.0) ;
```

```
}
```

5. Schritt: Erzeugung des Fragment-Shaders: Hier wird zuerst die Schnittstelle zum Vertex-Shader, die „varying“ Variable, deklariert. Mit dieser Variable, welche die aktuelle Position der Pixel bestimmt, kann in der Main-Methode die Pixel-Farbe berechnet und der Variablen „gl_FragColor“ zugewiesen werden. Wie beim Vertex-Shader können sowohl die Javascript-Textur als auch die definierten Parameter als „Uniforms“ eingebunden werden.

   Auch kann eine Fehlerkontrolle wie beim Vertex-Shader aktiviert werden.

6. Schritt: Falls weder Kompilierfehler beim Aktivieren der Checkboxen auftreten, noch Warnungen bei den Parametern angezeigt werden, kann die prozedurale Shader-Textur in der Datenbank oder per Link heruntergeladen werden. Wenn eine Speicherung in der Datenbank stattfindet, kann der Shader im UI-Elemente-Tab selektiert werden.

```
varying vec3 vPosition;
uniform float tips;
uniform float radius;
uniform float number;

float evaluate(float x, float y);
float _step (float i, float a);

void main() {
                    float color = evaluate(vPosition.x, vPosition.y);
                    gl_FragColor = vec4(color, color, color,0);
}

float evaluate(float x, float y) { ... }
```

3 Transfer von Referenzbild-Eigenschaften

In diesem Kapitel wird analysiert, wie Eigenschaften aus einem gegebenen Eingabebild extrahiert werden und die Parameter einer prozeduralen Textur so angepasst werden können, dass diese dem Referenzbild so ähnlich wie möglich sind.

3.1 Verwandte Arbeiten

Zuerst muss analysiert werden, welche Eigenschaften aus einem Referenzbild gewonnen werden können, denn durch den Vergleich der Eigenschaften der prozeduralen Textur und der Eigenschaften des Referenzbildes können die Textur-Parameter so angepasst werden, das die Eigenschaften und damit meistens das Aussehen sehr ähnlich wird. Eigenschaften können mit Hilfe von Bildstatistiken gemacht extrahiert werden. Im Paper „A Survey of Image Statistics Relevant to Computer Graphics“ [PCR11] werden aktuelle Bildstatistiken diskutiert. So wird zwischen Statistiken erster Ordnung, die nur einzelne Pixel betrachten, Statistiken zweiter Ordnung, die Pixel-Paare betrachten, und Statistiken höherer Ordnung unterschieden, die 3 oder mehr Pixel gleichzeitig betrachten. Bild-Statistiken erster Ordnung werden standardmäßig durch Histogramme dargestellt. Ein Histogramm ist eine graphische Häufigkeitsverteilung von Bildmerkmalen wie zum Beispiel Farben oder Helligkeit. Bild-Statistiken erster Ordnung sagen nicht viel über die räumliche Verteilung von Pixeln aus, da zwei komplett verschiedene Bilder dieselben Histogramme erzeugen können. Bildstatistiken erster Ordnung sind ausreichend, um Eigenschaften wie Bildhelligkeit, Kontrast oder Farbzusammensetzung zu vergleichen.

Zu den Bildstatistiken zweiter Ordnung zählen zum Beispiel die spektrale Leistungsdichte oder die Gradienten-Analyse. Die spektrale Leistungsdichte analysiert die relative Leistung von verschiedenen räumlichen Frequenzen. Mit Hilfe der Fourier-Analyse lassen sich zum Beispiel Kanten und Übergänge durch eine gewichtete Summe von Sinusschwingungen, bei der hohe Frequenzen weniger stark gewichtet werden, darstellen. Außerdem lassen sich durch eine spektrale Leistungsdichte-Analyse Regelmäßigkeiten in Bildern erkennen.

Die Gradienten-Analyse betrachtet die Beziehung von jeweils zwei Pixeln zueinander. Dadurch entsteht ein Gradienten Feld, in dem zum Beispiel homogene Oberflächen genauso wie scharfe Übergänge erkannt werden können.

Durch die Erweiterung zu einer Bildstatistik höherer Ordnung können Bildstrukturen noch besser analysiert werden. Dadurch, dass die Gradienten-Analyse mehr als zwei Pixel gleichzeitig betrachtet, können zum Beispiel Wellenstrukturen direkt erkannt werden.

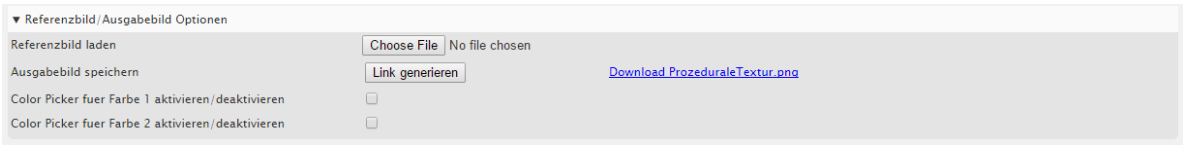

Abbildung 3.1: UI-Elemente für Referenz- und Ausgabebild aus der browserbasierten Entwicklungsumgebung: Laden und Anzeigen des Referenzbildes, Speichern des Textur-Ausgabebilds, Aktivieren der Color-Picker.

Um die Eigenschaften auf eine prozedurale Textur zu übertragen, müssen Parameter angepasst werden. Da Parameter selten direkt eine Struktur erstellen, wird im Paper „Interactive Parameter Retrieval for Two-Tone Procedural Textures “ [GKHF14] von L. Gieseke et al. ein Verfahren vorgestellt, in dem die Parameter automatisiert eingestellt werden und die Eigenschaften von Referenzbild und dem Bild der prozeduralen Textur verglichen werden. Die Prozess-Pipeline startet damit, dass aus dem Referenzbild eine „Color Blend Map“ sowie zwei Farben extrahiert werden. Zu der „Color Blend Map“ wird nun aus einer Datenbank von bereits generierten prozeduralen Textur-Bildern mit Hilfe einer Distanz-Metrik, welche die Ähnlichkeit von zwei verschiedenen Bildern beschreibt, eine passende prozedurale Textur mit Parametern ausgewählt. Durch Interpolation der Textur-Werte mit den zwei zuvor extrahierten Farben entsteht ein neues Bild, welches sich im optimalen Fall nur geringfügig vom Referenzbild unterscheidet.

An die Distanz-Metrik werden dabei sehr hohe Anforderungen gestellt: Zum einen sollen alle für den Benutzer wichtigen Merkmale im Referenzbild gefunden werden, ohne dass die Metrik Bildtyp spezifisch ist, zum anderen muss die Berechnung der Metrik in akzeptabler Zeit ablaufen.

3.2 Implementierte Interaktionsmöglichkeiten

Für die browserbasierte Entwicklungsumgebung steht kein automatisierter Algorithmus zur Verfügung, der die Parameter anpasst. Der Benutzer kann im UI-Elemente-Tab (Abbildung 3.1) ein Referenzbild laden und anzeigen lassen und mit Hilfe eines „Color Pickers“ zwei verschiedene Farben extrahieren, die zur Interpolation der Pixelfarben der prozeduralen Javascript-Textur verwendet werden. Weitere Interaktionsmöglichkeiten, die in die Browserbasierte Entwicklungsumgebung für prozedurale Texturen beliebig integriert werden können und vom Interaktiven Aufbau der Webseite stark profitieren, müssen noch implementiert werden. Der Benutzer kann allerdings bereits die Parameter der Textur beliebig anpassen und das durch die prozedurale Textur erzeugte Bild manuell speichern. Ein Vergleich von Referenzbild mit Textur-Bild durch ein externes Programm ist deshalb möglich.

4 Aufzeichnen von Benutzer-Interaktionen

Die Menge an Programmen und Webseiten, die bestimmte Aufgaben erfüllen, steigt immer weiter. Welches Programm oder welche Webseite vom Benutzer ausgewählt wird, ist von verschiedenen Faktoren abhängig. Ein wichtiger Faktor ist die Qualität, da diese bestimmt, wie effizient und einfach ein Programm oder eine Webseite benutzt werden kann. Um zum Beispiel die Bedienbarkeit von Webseiten zu verbessern, ist es hilfreich herauszufinden, wie Benutzer mit dieser Webseite interagieren, um nach einer genauen Analyse Verbesserungen auszuarbeiten. Viele Programme und Webseiten speichern nur Daten; wie diese Daten eingegeben werden, wird nicht gespeichert.

Für die browserbasierte Entwicklungsumgebung ist die Frage wichtig: „Wie werden Daten eingegeben oder wie werden Parameter verändert, um die prozedurale Textur an das Referenzbild anzupassen?“. Denn damit kann die Bedienbarkeit der browserbasierten Entwicklungsumgebung verbessert werden und es können Rückschlüsse auf die automatische Anpassung von Parametern erlangt werden.

4.1 Verwandte Arbeiten

Um mehr über das Aufzeichnen von Benutzerinteraktionen herauszufinden, wird das Paper “Knowing the User’s Every Move – User Activity Tracking for Website Usability Evaluation and Implicit Interaction„ [AWS06] betrachtet, welches analysiert, wie ein genaues Aufzeichnen von Benutzer Interaktionen funktioniert, ohne dass die Benutzererfahrung negativ beeinflusst wird. Ein wichtiger Punkt ist, der im Paper beschrieben wird, dass ein Aktivitäts-Tracking-Ansatz, der direkt auf dem Server Daten sammelt, bei vielen Webseiten nicht mehr funktioniert. Das Server-Client-Konzept, bei dem der Client nur als Eingabe- und Darstellungsgerät fungiert und Berechnungen und Speicherungen auf dem Server ausgeführt werden, wird immer öfter durch ein neues Konzept abgelöst, bei dem der Client mittels Javascript die Berechnung und Vorverarbeitung von Eingabedaten und in seltenen Fällen durch eine lokale Datenbank auch die Speicherung der Daten übernimmt. Der Server ist in diesem Fall nur dazu da, die Webseite zu laden und Daten zu laden oder speichern.

Der Ansatz, der im Paper vorgestellt wird, benutzt einen Proxy, der zwischen Server und Client positioniert wird und die Daten der Webseite modifiziert, bevor sie auf dem Webbrowser des Clients ausgeführt werden. Dabei wird Javascript-Code an die Webseite angehängt, der die Funktionalität für das Aufzeichnen von Benutzerinteraktionen übernimmt und eine Log-Datei generiert. Das bietet die Möglichkeit, vom Server unabhängig ein Aktivitäts-Tracking durchzuführen.

Es gibt verschiedene Möglichkeiten, die Aktivität von Benutzern mit Webseiten aufzuzeichnen. Durch das Speichern von Mausinteraktionen, wie Bewegung, Rechtsklick, Linksklick, Mittelklick, Scrollen und Tastatureingaben können viele Informationen gewonnen werden. Zusätzlich kann durch eine externe Kamera die Bewegung der Augen und somit der Blickfokus gespeichert werden. Das erfordert

externe Programme und Hardware und ist deshalb aufwändiger. Im Paper „What can a mouse cursor tell us more? Correlation of eye/mouse movements on web browsing.“ [CAS01] wird darauf hingewiesen, dass ein starker Zusammenhang zwischen Mausbewegung und Blickfokus besteht. Somit kann in vielen Fällen auf „Eye-Tracking“ verzichtet werden, um Kosten zu sparen und das Aufzeichnen von Benutzerinteraktionen ohne externe Hardware und Software durchzuführen. Bei der browserbasierten Entwicklungsumgebung für prozedurale Texturen könnte „Eye-Tracking“ dennoch wichtig sein, um Rückschlüsse auf eine automatisierte Einstellung der Parameter zu bekommen. Dadurch, dass die Maus mit dem Einstellen der Parameter beschäftigt ist, während die Augen hauptsächlich das Referenzbild und die Bildausgabe der prozeduralen Textur betrachten, kommt es zu großen Abweichungen zwischen der Mausposition und dem Blickfokus. Mit Hilfe von „Eye-Tracking“ könnten auch die für den Benutzer wichtigen Merkmale des Referenzbildes gewonnen werden.

Gespeicherte Informationen über Benutzerinteraktionen sind nicht sehr abstrakt. Es ist zum Beispiel nur bekannt, dass die Maus einen Klick auf Position (x,y) ausgeführt hat. Um sinnvolle Aussagen über die Benutzerinteraktionen zu erstellen, müssen die gespeicherten Informationen veranschaulicht werden. Dazu ist Wissen über das Webseitenlayout notwendig, da sonst einem Mausklick keine Aktion wie zum Beispiel Speichern oder Laden zugewiesen werden kann. Bei hoch dynamischen Webseiten, die eine Tab-Struktur besitzen und dynamische Eingabeelemente, muss der aktuelle Zustand einer Webseite entweder direkt in der Log-Datei oder bei der Veranschaulichung berechnet werden, um eindeutige Aussagen über die verwendeten Aktionen zu treffen.

Nach der Veranschaulichung können durch die Analyse verschiedene Informationen gewonnen werden: Zum einen, welche Aktionen in welcher Reihenfolge ein Benutzer vorgenommen hat. Diese Information ist zum Beispiel relevant für die Frage wie ein Benutzer die Parameter einer prozeduralen Textur einstellt. Durch die Analyse der gesammelten Informationen lassen sich auch Rückschlüsse auf den Benutzer erstellen: Person A liest die Anleitung nicht richtig und führt falsche Aktionen aus. Zum Anderen lassen sich Aussagen über die Präzision beim Klicken oder die Geschwindigkeit beim Tippen feststellen. Durch die Analyse können genauso gut kritische Informationen über die Webseite gefunden werden: Ein häufiges Daneben-Klicken der Benutzer schließt auf zu kleine Buttons oder das Finden einer Option, die häufig benötigt wird, braucht zu lange und deshalb sollte diese spezifische Option besser positioniert sein.

4.2 Implementierung

Für die browserbasierte Entwicklungsumgebung ist ein detailliertes Tracking von Benutzerinteraktionen implementiert. Um dieses zu aktivieren, kann der Benutzer im UI-Elemente Tab (Abbildung 4.1) eine Checkbox aktivieren. Dann werden automatisch die Mausbewegung, Mausklicks, Mausscrollen und Tasteneingaben in ein Logfile gespeichert, das heruntergeladen werden kann.

Im Gegensatz zum oben genannten Paper wird hier allerdings kein Proxy programmiert, der auch dann noch Benutzerinteraktionen aufzeichnet, wenn der Benutzer die Webseite verlassen hat, da für die browserbasierte Entwicklungsumgebung nur direkte Interaktionen mit der Webseite wichtig sind. Zum Beispiel ist es interessant herauszufinden, wie ein Benutzer die Parameter einstellt; ob er hingegen noch weitere Webseiten währenddessen, besucht ist nicht relevant.

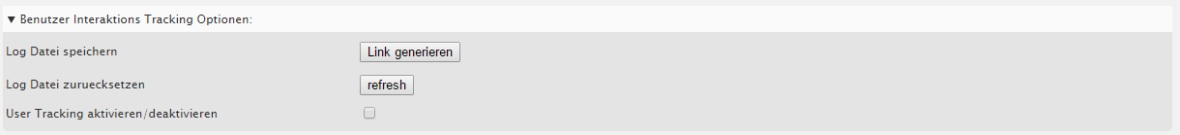

Abbildung 4.1: UI-Elemente für das Benutzerinteraktions-Tracking aus der browserbasierten Entwicklungsumgebung: Log-Datei speichern, Log-Datei zurücksetzen, Benutzerinteraktions-Tracking aktivieren/deaktivieren.

Listing 4.1 Beispiel für JSON-Objekt eines Tastatur-Taste-Drücken Events: In „time“ steht der Zeitstempel, in „option“ der Auslöser des Events, „position“ beinhaltet, falls relevant, die Mausposition, und „message“, welche Taste gedrückt wurde.

```
{
    "time": "2024-10-15 23:59:58.587",
    "option": "onkeydown",
    "position": "(0,0)",
    "message": "U+0053"
},
```

Die Implementierung der Tracking-Funktion benutzt verschiedene Javascript Events:

- Maustaste drücken und loslassen
- Mausbewegung
- Mausraddrehen
- Tastatur-Taste drücken und loslassen

Die Mausbewegung wird exakt gespeichert. Damit Informationen über Klicks und Tastatureingaben in der Log-Datei direkt erkennbar sind, wird der Datenstrom unterteilt in Events ohne Mausbewegung und in einen Teil, in dem alle Events vorhanden sind. Die Events werden als JSON Objekte mit den Attributen „time, option, position, message“ abgespeichert. Ein Programm, das diese Daten auswertet, kann direkt mit diesen Objekten rechnen. Standardmäßig ist das Benutzerinteraktion-Aufzeichnen ausgeschaltet, da sonst innerhalb von kurzer Zeit viel Information gespeichert werden muss. Für eine Benutzerstudie kann diese Funktion jederzeit aktiviert werden.

4.3 Anforderungen an eine Benutzerstudie

Um eine aussagekräftige Benutzerstudie durchzuführen, sind Schritte notwendig, die hier aufgelistet werden. Nimmt man an, das Ziel einer Studie sei das Finden von Rückschlüssen zu einer automatischen Anpassung von Parametern einer prozeduralen Textur, um ein Referenzbild nachzubilden, so ist es sinnvoll, dass die prozedurale Textur und die Referenzbilder vom Studien-Betreuer ausgewählt werden. Sonst wäre das Ziel nicht die automatische Anpassung von Parametern, sondern auch das Finden einer passenden prozeduralen Textur. Wenn die Anzahl der Ziele einer Studie steigen, kann die Aussagekraft

der Studie fast genauso schnell sinken. Außerdem ist das Anpassen von Parametern ohne großes Fachwissen möglich. Für das Auswählen der passenden Textur ist Vorwissen über gegebene Texturen notwendig.

Wenn ein Referenzbild oder mehrere Referenzbilder und die dazugehörigen prozeduralen Texturen ausgewählt sind, müssen die Studienteilnehmer in die Aufgabenstellung eingewiesen werden. Die ist in diesem Fall das Reproduzieren des Referenzbildes mit der prozeduralen Textur, indem Parameter verändert werden. Es ist außerdem sinnvoll, die Aufgaben zu unterteilen, um bessere Vergleiche zwischen den Studienteilnehmern ziehen zu können. Zum Beispiel könnte ein Aufgabenteil das Auswählen der Farben sein oder das Einstellen einer Gruppe von Parametern, die für einen Teil der Bild Eigenschaften zuständig sind.

Des weiteren sollten die Teilnehmer über die Funktionalität der „Color Picker“ und anderen Elementen der Webseite unterrichtet werden.

Ein weiterer Schritt ist entweder das Modifizieren der Entwicklungsumgebung, sodass zum Beispiel der Studienteilnehmer keine Möglichkeit hat, die Aufzeichnung der Benutzerinteraktion zu deaktivieren und die Log-Datei zu löschen, oder es werden Regeln für das Benutzen der browserbasierten Entwicklungsumgebung aufgestellt, an die sich die Teilnehmer halten müssen.

Sobald die Webseite und die Teilnehmer ausreichend vorbereitet sind, das heißt unter anderem auch das Einstellen der prozeduralen Textur und des Referenzbildes, kann die Studie beginnen.

Nachdem die Teilnehmer die Aufgaben abgearbeitet haben, ist es notwendig, die Log-Dateien sowie die Ausgabebilder abzuspeichern. Außerdem kann für zusätzliches Feedback eine Teilnehmerbefragung ausgeführt werden.

Die Log-Dateien müssen verarbeitet und veranschaulicht werden. Dazu ist es notwendig, Programme zu schreiben, die diesen Prozess übernehmen. Zusätzlich zur Log-Datei kann auch die Qualität des Ausgabebildes, das der Studienteilnehmer durch Einstellen der Parameter erzeugt hat, analysiert werden.

5 Zusammenfassung und Ausblick

In dieser Arbeit wird analysiert, wie Texturen erstellt werden können und wie der Benutzer beim Erzeugen von prozeduralen Texturen durch eine Entwicklungsumgebung unterstützt werden kann. Es wird eine browserbasierte Entwicklungsumgebung vorgestellt, die sowohl das Programmieren von prozeduralen Javascript-Texturen, die auf der CPU berechnet werden, als auch das Programmieren von Shadern ermöglicht, die auf der GPU berechnet werden. Im Teil wird analysiert, wie aus einem Referenzbild eine prozedurale Textur erstellt werden kann und welche Schritte und Methoden dafür notwendig sind. Im dritten Teil der Arbeit wird analysiert, wieso das Aufzeichnen von Benutzerinteraktionen wichtig für die Qualität einer Webseite ist und wie damit Rückschlüsse auf eine automatisierte Erzeugung von Parametern möglich werden. Es wird außerdem beschrieben, wie das Aufzeichnen von Benutzerinteraktionen für die browserbasierte Entwicklungsumgebung realisiert wurde und welche Schritte notwendig sind, um eine Benutzerstudie durchzuführen.

Für die browserbasierte Entwicklungsumgebung lassen sich weitere Features entwickeln, die dem Benutzer mehr und bessere Möglichkeiten bieten, prozedurale Texturen zu erstellen. Ein Teil der Features kann als Input-Features klassifiziert werden. Das heißt zum Beispiel, dass es dem Benutzer ermöglicht wird, für den Vertex- und Fragment-Shader Bilder als Eingabetexturen, oder Film-, Maus- und Tastatur-Eingaben zu verwenden, wie zum Beispiel auf der Webseite Shadertoy begrenzt angeboten wird.

Auch die dynamisch erstellten Parameter, die bisher auf Floats begrenzt sind, kann man erweitern, sodass auch dynamisch definierte Farb-Parameter möglich sind. In der aktuellen Version arbeitet die browserbasierte Entwicklungsumgebung im WebGL-Fenster nur mit einem einfachen Plane. Hierfür wäre eine Auswahl an 3D-Objekten sinnvoll oder sogar die Möglichkeit, 3D-Objekte zu importieren, die in 3D-Entwicklungsumgebungen wie Blender [Ble14] oder Maya [may] entwickelt wurden.

Auch die Editor-Umgebungen können noch ausgebaut werden. Die Möglichkeit, Hilfsfunktionen getrennt von der Hauptfunktion in die Datenbank zu laden und zu speichern, wäre hilfreich oder generell das Einführen von Hilfsfunktionen für die Shader-Umgebung.

Ein weiteres Feature ist das Erstellen einer Online-Datenbank, in der prozedurale Texturen für die browserbasierte Entwicklungsfunktion verfügbar gemacht werden.

Bisher sind wenige Interaktions-Möglichkeiten mit dem Referenzbild bereitgestellt. Für das Eingabe- und Ausgabebild könnten Statistiken berechnet und angezeigt werden, sowie eine Metrik, welche die Ähnlichkeit beider Bilder bestimmt. Ob die Erzeugung einer Datenbank mit gerenderten prozeduralen Texturen als Browser-Anwendung effizient funktioniert, um die Parameter automatisiert einzustellen, müsste getestet werden.

Als wichtigster Ausblick bietet sich die Möglichkeit, eine Studie durchzuführen, wie sie in Abschnitt 4.3 beschrieben ist. Durch die Studie könnten zum einen weitere Verbesserungen für die browserbasierte Entwicklungsumgebung herausgefunden werden und zum anderen Rückschlüsse auf eine automatisierte Anpassung von Textur-Parametern um ein Referenzbild durch eine prozedurale Textur nachzubilden.

Programme, welche die Log-Dateien, die von der browserbasierten Entwicklungsumgebung erstellt werden, auswerten und veranschaulichen, müssen allerdings noch erstellt werden. Dabei ist es notwendig die einfachen Events der Log-Datei zu interpretieren, das heißt das aus einem Klick an Position (x,y) eine genauere Aktion wird, wie zum Beispiel das Drücken des Lade-Buttons.

Literaturverzeichnis

[AWS06] R. Atterer, M. Wnuk, A. Schmidt. Knowing the User's Every Move: User Activity Tracking for Website Usability Evaluation and Implicit Interaction. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, S. 203–212. ACM, New York, NY, USA, 2006. doi:10.1145/1135777.1135811. URL http://doi.acm.org/10.1145/1135777.1135811. (Zitiert auf Seite 37)

[Ble14] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam, 2014. URL http://www.blender.org. Zugriff: 14-November-2014. (Zitiert auf den Seiten 22 und 41)

[Bom] GPU Gems - Chapter 20. Texture Bombing. http://http.developer.nvidia.com/GPUGems/gpugems_ch20.html. Zugriff: 14-November-2014. (Zitiert auf den Seiten 6 und 17)

[CAS01] M. C. Chen, J. R. Anderson, M. H. Sohn. What Can a Mouse Cursor Tell Us More?: Correlation of Eye/Mouse Movements on Web Browsing. In *CHI '01 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '01, S. 281–282. ACM, New York, NY, USA, 2001. doi:10.1145/634067.634234. URL http://doi.acm.org/10.1145/634067.634234. (Zitiert auf Seite 38)

[DCr] Top Down Operator Precedence. http://javascript.crockford.com/tdop/tdop.html. Zugriff: 14-November-2014. (Zitiert auf Seite 28)

[GDG12] G. Gilet, J.-M. Dischler, D. Ghazanfarpour. Multi-scale Assemblage for Procedural Texturing. *Computer Graphics Forum*, 31(7):2117–2126, 2012. doi:10.1111/j.1467-8659.2012.03204.x. URL http://dx.doi.org/10.1111/j.1467-8659.2012.03204.x. (Zitiert auf den Seiten 6, 15 und 16)

[GKHF14] L. Gieseke, S. Koch, J.-U. Hahn, M. Fuchs. Interactive Parameter Retrieval for Two-Tone Procedural Textures. *Computer Graphics Forum*, 33(4):71–79, 2014. doi:10.1111/cgf.12414. URL http://dx.doi.org/10.1111/cgf.12414. (Zitiert auf den Seiten 21 und 36)

[GLS] GLSL Sandbox Gallery. http://glslsandbox.com/. Zugriff: 14-November-2014. (Zitiert auf den Seiten 6, 9, 19 und 20)

[Imp] Improved Noise reference implementation. http://mrl.nyu.edu/~perlin/noise/. Zugriff: 14-November-2014. (Zitiert auf den Seiten 6, 14 und 21)

[ind] Indexeddb - Web API Interfaces MDN. https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API. Zugriff: 14-November-2014. (Zitiert auf Seite 24)

[KHR] KHRONOS GROUP. Webgl specification. editor's draft. https://www.khronos.org/registry/webgl/specs/latest/1.0/. Zugriff: 14-November-2014. (Zitiert auf Seite 21)

[may] Maya Autodesk Comprehensive 3D animation software. http://www.autodesk.com/products/maya/overview. Zugriff: 14-November-2014. (Zitiert auf den Seiten 22 und 41)

[PCR11] T. Pouli, D. W. Cunningham, E. Reinhard. A Survey of Image Statistics Relevant to Computer Graphics. *Computer Graphics Forum*, 30(6):1761–1788, 2011. doi:10.1111/j.1467-8659.2011.01900.x. URL http://dx.doi.org/10.1111/j.1467-8659.2011.01900.x. (Zitiert auf Seite 35)

[Per85] K. Perlin. An Image Synthesizer. *SIGGRAPH Comput. Graph.*, 19(3):287–296, 1985. doi:10.1145/325165.325247. URL http://doi.acm.org/10.1145/325165.325247. (Zitiert auf den Seiten 14 und 15)

[Pra73] V. R. Pratt. Top Down Operator Precedence. In *Proceedings of the 1st Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '73, S. 41–51. ACM, New York, NY, USA, 1973. doi:10.1145/512927.512931. URL http://doi.acm.org/10.1145/512927.512931. (Zitiert auf Seite 28)

[Sha] Shadertoy BETA. https://www.shadertoy.com/. Zugriff: 14-November-2014. (Zitiert auf den Seiten 6, 9, 17 und 18)

[THR] three.js Javascript 3D library. http://threejs.org/. Zugriff: 14-November-2014. (Zitiert auf Seite 21)

[Web] WebGL Reference Card. https://www.khronos.org/files/webgl/webgl-reference-card-1_0.pdf. Zugriff: 14-November-2014. (Zitiert auf Seite 29)

[WLKT09] L.-Y. Wei, S. Lefebvre, V. Kwatra, G. Turk. State of the Art in Example-based Texture Synthesis. In *Eurographics 2009, State of the Art Report, EG-STAR*. Eurographics Association, 2009. URL http://www-sop.inria.fr/reves/Basilic/2009/WLKT09. (Zitiert auf Seite 14)

[Wor96] S. Worley. A Cellular Texture Basis Function. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, S. 291–294. ACM, New York, NY, USA, 1996. doi:10.1145/237170.237267. URL http://doi.acm.org/10.1145/237170.237267. (Zitiert auf den Seiten 6 und 16)

[wsc] JavaScript Math Reference. http://www.w3schools.com/jsref/jsref_obj_math.asp. Zugriff: 14-November-2014. (Zitiert auf Seite 28)

Alle URLs wurden zuletzt am 14. 11. 2014 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift