Fachhochschule
Kaiserslautern
University of Applied Sciences

Informatik und
Mikrosystemtechnik
Zweibrücken

Studiengang Medieninformatik

Multimediaprojekt Phase II

Wintersemester 2012/13

Natural Interfaces mit VRPN

von

Sascha Hayton, 861309
Patrick Gab, 862108

Betreut von Prof. Brill

11. Februar 2013

Inhaltsverzeichnis

1 Einleitung

1.1 Zieldefinition

Ziel des Projektes ist es eine Anwendung weiter zu entwickeln, die bereits von uns im Sommersemester 2012 in der Veranstaltung Grafikprogrammierung erstellt wurde.

Dabei handelte es sich um eine „Virtuelle Graffiti“ Applikation. Es sollte dem Benutzer ermöglicht werden auf einer Wand zu malen. Es sollte VRPN und OpenGL zum Einsatz kommen.

1.2 Kurzbeschreibung

Nach Überlegungen und Rücksprache mit Prof. Brill, kamen wir dann zu folgenden Anforderungen, die wir realisieren wollen.

- Datenstruktur der Anwendung ändern
  - Vererbung; Aufbau der Subjekte, Linien, Formen
  - Sauberes MVC implentieren; Es soll möglich sein den View für eine Linie aus Tastendruck auszutauschen.
- Einbau eines GLSL- Shader, der für die Darstellung einer Linie beeinflussen kann
- Neue View für Linien
- Quellcode „säubern“ (z.B. unverständliche Kommentare und Konsolenausgaben entfernen)
- Die Breite einer Line soll von der Entfernung zum Tracker abhängig sein.
- Blending mehrere aufeinanderliegender Linien.
- Überarbeitung des alten Projektberichts

Als Anregung nutzten wir eine Flash Anwendung die unter folgender URL zu erreichen ist:
http://www.totalgraffiti.com/

1.3 Allgemeine Bemerkungen

Dieser Berichtsoll nur einen Überblick darüber geben, was die Hintergedanken waren und welchen Zusammenhang bestimmte Klassen haben.

Wir davon aus, dass Grundkenntnisse in C++, Computergrafik vorhanden sind. Außerdem ist es von Vorteil wenn das MVC-Entwurfsmuster bekannt ist.

1.4 Begriffserklärung

Anwendungsklasse:
Wenn von der Anwendungsklasse die Rede ist, dann ist die Klasse GraffitiEngine gemeint.

VRPN „Virtual Reality Peripheral Network“:
[1]*„VRPN ist ein generisches System mit dem möglichst schnell neue und innovative Peripherie-Geräte an Anwendungen gehängt werden können“*.

FAAST „Felexible Action and Articulated Skeleton Toolkit“:
FAAST ist das Programm, dass für die Kinect verantwortlich ist. FAAST enthält ein VRPN Server.

GLSL „OpenGL Shading Language“:
GLSL ist eine Programmiersprache um auf der Grafikkarte eigene Programme, sogenannte Shader auszuführen.

Git:
Git ist eine kostenlose und offene Versionsmanagementsoftware.

2 Planung

2.1 Allgemeines

Zu Beginn hatten wir es uns so vorgestellt, dass wir genau so weiter machen wie wir im Sommer Semester 2012 aufgehört haben; keine Versionsverwaltungssoftware und immer über Dropbox Quellcode austauschen. Das hat letztes Mal auch gut funktioniert, das wir uns oft im VR-Labor getroffen haben und dort viel gearbeitet haben.

Dieses Mal jedoch haben wir uns dazu entschieden mit GIT zu arbeiten. Zu dieser Entscheidung kamen wir nach dem wir festgestellt haben, dass wir in diesem Semester wohl weniger Zeit für die Treffen haben werden und das VR-Labor mit anderen Studenten teilen müssen, was für uns bedeutet hat, dass wir wohl mehr von zuhause aus arbeiten müssen.

2.2 Anwendung

Die Anwendung wurde im MVC-Modell gebaut, da es sich besonders anbietet die Darstellung, Steuerung und unterschiedliche Zeichnungen getrennt voneinander zu betrachten.

Der Controller (Steuerung) wertet die eintreffenden Benutzeraktionen aus und leitet sie weiter. Das Model (Zeichnungen) enthält verschiedene Informationen, die benötigt werden um das Model im View darstellen zu können. Der View (Darstellung) bekommt eine Liste von Subjekt-Objekten, die er dann darstellt.

Momentan besteht die Anwendung aus einer Anwendungsklasse, die als Controller dient, aus mehreren View-Klassen und mehreren Modell-Klassen.

Zu Beginn wurde ein Klassendiagramm in dem festgelegt wurde die Vererbung später aussehen soll. (*Entwurf eines Klassendiagram für geplante Vererbung*).

3 Umsetzung

3.1 Die Anwendungsklasse

Die Klasse GraffitiEngine ist von der Klasse vlgVRPNEngine abgeleitet und ist die Anwendungsklasse. Sie dient als Controller unsers Programmes. In dieser Klasse wird der OpenGL Kontext aufgebaut, die VRPN-Geräte eingestellt und die einkommenden Daten verarbeitet und weitergeleitet. Auch die „ActionListener“ befinden sich in dieser Klasse.

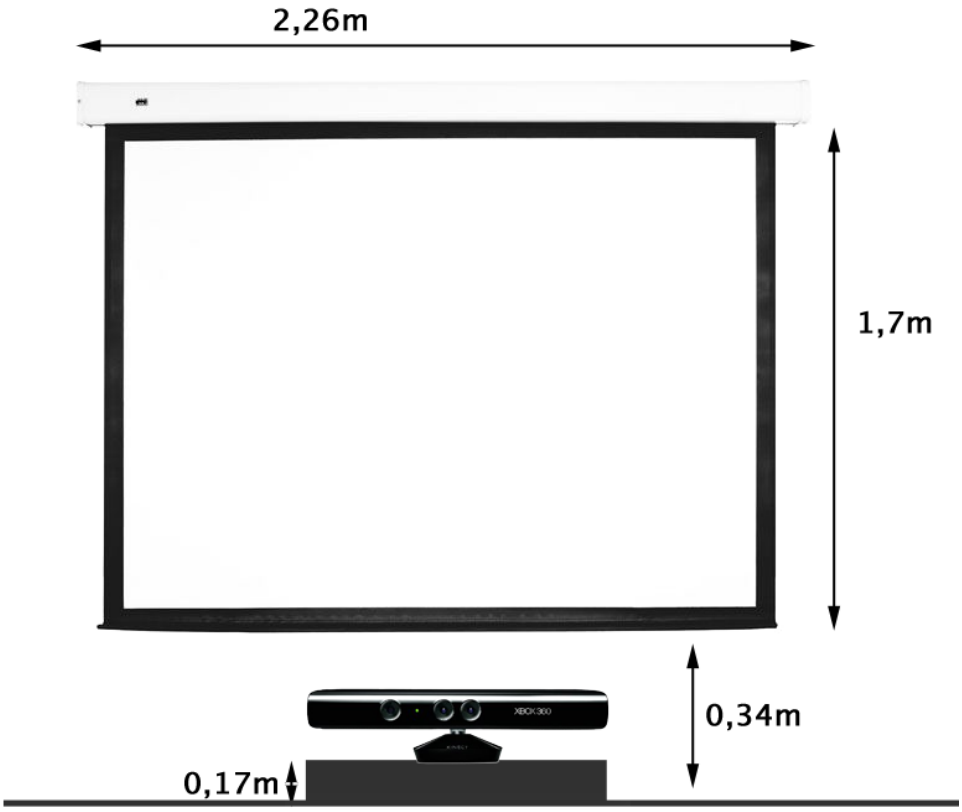
3.2 Models und Views

Die Modelle unserer Anwendung sind alle von der Klasse vlgSubject abgeleitet. Sie enthalten alle Attribute die gebraucht werden damit sie im View dargestellt werden können Zu jedem Modell gibt es eine oder mehrere Views In unserer Anwendung gibt es Folgende Modelle und Views:

Modell	View	Erläuterung
Line	CircleLineView PolyLineView	Diese Klassen sind für die Ausgabe einer Linie verantwortlich
Wall	WallView	Die Klasse Wall ist für das Interface und den Hintergrund der Anwendung verantwortlich. Zum Interface zählt ein Cursor der die momentane Sprühposition, und die verwendete Farbe anzeigt und ein Symbol, das anzeigt, ob gerade Daten vom Tracker empfangen werden oder nicht.
Circle	CircleView	Diese beiden Klassen sind für die Ausgabe eines Kreises verantwortlich
Triangle	TriangleView	Diese beiden Klassen sind für die Ausgabe eines Dreieckes verantwortlich
Stamp	StampView	Diese beiden Klassen sind dafür da eine Textur auf die Leinwand zu kleben. Diese Klasse wurde noch nicht optimiert und befindet sich in einem prototypartigen Status. Das bedeutet, sie funktionieren, aber um neue Texturen hinzuzufügen muss an mehreren Stellen der Code angefasst werden, was nicht sehr optimal ist.

3.3 Leinwand und Koordinatentransformation

Als Sprühgrund dient die Leinwand mit Rückprojektion im VR-Labor.
Anmerkung: Die Kinect kann interacticve geneigt werden! Letzter Stand war 15°.



Unser Ziel war es die Daten die wir von der Kinect bekamen in für uns brauchbare Bildschirmkoordinaten um zurechnen. Dabei gingen wir von einem 4:3 Format aus.
Analog zu der Umrechnung der Kinect-Daten, haben wir auch eine Umrechnung von Maus-Daten.

Vorgegangen sind wir wie folgt:

1. Pitch-Winkel der Kinect kompensieren
    a. Rotation rückgängig machen:

    $$y_{neu} = \cos(pitch) * y_{alt} + \sin(pitch) * y_{alt}$$

    Ergebnis: Y-Wert bzgl. Kinect, aber ohne Pitch.

2. Koordinatenverschiebung
    Wir wollen die Koordinaten die wir bekommen so verändern, so dass der Ursprung in der Mitte der Leinwand ist. Der Grund dafür ist, dass unsere orthogonale Projektion auch einen solchen Ursprung hat.

    $$y_{new} = y_{alt} - \left(\frac{LeinwandHoehe}{2}\right)$$

$$y_{new} = y_{alt} - \frac{1{,}7m}{2}$$

$$y_{new} = y_{alt} - 0{,}85m$$

3. Umrechnung in Bild Koordinaten
   Wir wollen die Koordinaten so umrechnen, so dass sie Bildkoordinaten entsprechen. Dazu verwenden wir eine Abbildung:

$$Abbildung: [-1{,}13\,;1{,}13] \rightarrow [,-4\,;\,4]$$

$$x: -4 + \frac{x + 1{,}13}{2{,}26} * 8$$

$$Abbildung: [-0{,}65\,;0{,}65] \rightarrow [-3\,;3]$$

$$y: -3 + \frac{y + 0{,}65}{1{,}7} * 6$$

$$y_{new} \rightarrow \cos(pitch) * y_{alt} + \sin(pitch) + z_{alt} - 0{,}85$$

$$x_{new} \rightarrow -4 + \frac{x_{alt} + 1{,}13}{2{,}26} * 8$$

$$y_{new} \rightarrow -3 + \frac{y_{new} + 0{,}65}{1{,}7} * 6$$

Anmerkung: Diese hergeleiteten Koordinatentransformationen wurden in gemeinsamer Arbeit mit Prof. Brill ausgearbeitet.

1.1.1 Anwendung im Code

Implementiert wurden die Formeln in einer Funktion schon letztes Semester, uns ist allerdings ein Fehler aufgefallen und nach einem Ratschlag eines Master-Studenten ist uns eine Idee gekommen wie wir die Funktion verbessern und optimieren können.

GraffitiEngine.cpp

```
void GraffitiEngine::coordAdju(float coord[]){
     //   return ((cosPitch*wY + sinPitch*wZ) -0.82f);
     float  oldX = coord[0];
     float  oldY = coord[1];
     float  oldZ = coord[2];
     coord[0] = (-4.0f + ((oldX+1.13f)/2.26f) *8.0f);
     coord[1]  =  ((-3.0f  +  ((cosPitch*oldY  +  sinPitch  *oldZ)+
0.65)/ 1.7f) *6.0f);
     coord[2] = -sinPitch *oldY + cosPitch *oldZ;
}...
```

3.4 Line und ihre View

Hier werden wir auf die in *3.2 Models und Views* bereits erwähnte Line etwas genauer eingehen. Die Klasse Line enthält unter anderem drei Listen<float>. In diesen Listen werden die einkommenden X-, Y- und Z-Werte gespeichert. Die Anwendungsklasse besitzt eine lineList<Line>. In dieser lineList <Line> wird im Konstruktor eine leere Line hinzugefügt. Sobald vom Benutzer der Sprüh Knopf gedrückt wird, wird diese leere Line mit Werten gefüllt. Da diese lineList<Line> einem Observer zugewiesen ist, wird automatisch die draw() Methode der View aufgerufen.

GraffitiEngine.cpp

```
public GraffitiEngine(void) : ...{
tLine = new Line(colorIndex);
lineList.push_back(*tLine);
...
}
...
void handleAnalog(void*userData, const vrpn_ANALOGCB a){
...
if(pressed){
     lineList.front().myX.push_back(trackAry[0]);
     lineList.front().myY.push_back(trackAry[1]);
     lineList.front().myZ.push_back(trackAry[2]);
     lineList.front().myUndoSizeZ = zUndoSize;
     lineList.front().notify();
}...
```

Sobald die Sprühtaste losgelassen wird, wird diese Line die sich momentan an lineList[0] befindet um eine Stelle weiter, also auf lineList[1] geschoben. LineList[0] wird wieder geleert. Die View-Klasse bekommt als Parameter diese lineList<Line>. Es wird durch diese Liste durch gegangen und die jeweiligen Lines gemalt.

CircleLineView.cpp

```
...
//Schleife durch alle Linien
for(int i = 0;i<line.size();i++){
if (i == 0) zE = 2; //Position in der Z-Ebene; 2 bedeutet vorne
else zE = (float)(line.at(i).getMyUndoSizeZ()*0.02f);

/* myUndoSize gibt um das wievielte gemalte Objekt es sich han-
delt Hat man zum Beispiel zwei Lines gemalt so ist die myUndoSize
der ersten Linie eins und die der zweiten Linie zwei.
Dies hat den Effekt, dass das aktuell gemalte Objekt immer im
Vordergrund ist.
*/
     //Schleife durch alle Punkte einer Linie
     for(int j = 0; j< line.at(i).myX.size();j++){
     ...
          //Schleife um einen Kreis zu malen
          for (int k=32; k>=0; k--) {...}
     }
```

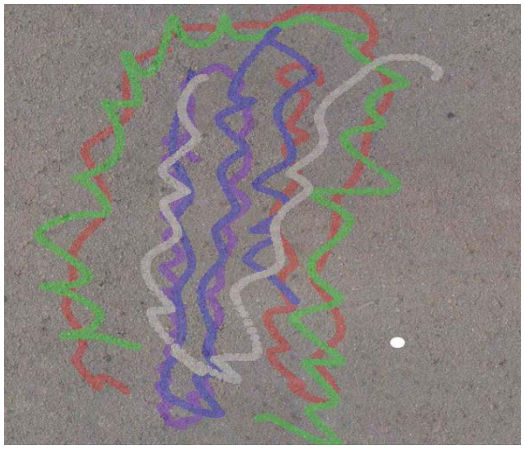
3.5 Gesten

Die WiiMote befindet sich in der rechten Hand und dient als Sprühdose, somit war klar die Daten der rechten Hand für die Anwendung zu wählen. Das Malen der Polygonzüge, sowie die Bitmaps und Formen richten sich an die Tracker-Daten der rechten Hand. Um nun per Wischen Geste den Hintergrund auszutauschen, wird vom Tracker der Abstand der linken Hand und linken Schulter als Ansatzpunkt zum Beginn der Geste gewählt. Sobald diese Voraussetzung erfüllt ist, prüft die Anwendung den Zurückgelegten Weg der linken Hand,

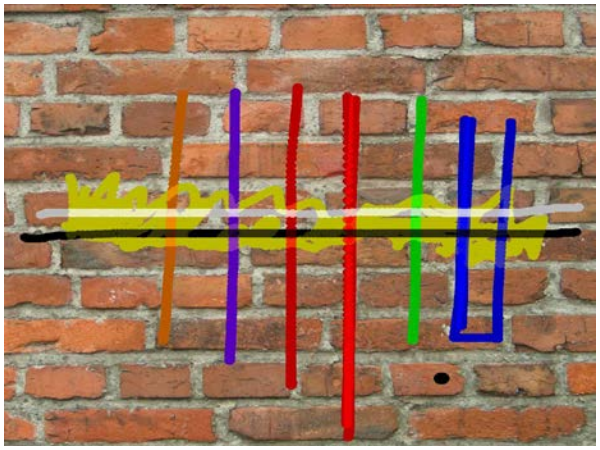
überschreitet jener einen zuvor bestimmten Wert und die Startvoraussetzung ist weiterhin erfüllt, wird die Wischen Geste aktiviert. Analog kann mit dem Wischen in die entgegengesetzte Richtung der zu vorgewählte Hintergrund ausgewählt werden. Der FAAST-Server ermöglicht das Einstellen eines Mirror-Mode, dieser Ermöglicht es uns die Anwendung für Rechts-, als auch Linkshänder individuell anzupassen. Die Logik der Anwendung ist hier nicht betroffen und funktioniert fehlerlos, mit dem Unterschied intern die rechte und linke Hand, sowie Schulter-Daten zu tauschen.

3.6 Blending und GLSL

Durch die Benutzung von GLSL-Shadern und Blending der Alpha-Werte der Farben erhofften wir uns, dass die Farben einen wirklichen „Sprüh“ Effekt bekommen und das wenn mehrere Farben aus einander Liegen, diese sich verändern, also sich mischen. Anfangs haben wir mit dem Befehl discard versucht, nur jeden dritten oder vierten Pixel zu malen um eine Transparenz vorzutäuschen. Allerdings waren wir mit dem Ergebnis nicht zufrieden.



Da es keinen Unterschied machte ob man mehrfach über die gleiche Stelle malte. Danach versuchten wir mit Blending der Alpha-Werte und einem GLSL-Shader, der den Pixel einer Line unterschiedliche Alpha-Werter zuweist, einen besseren Effekt zu erzeugen. Allerdings hat auch dieser Versuch uns nicht zufrieden gestellt.



Leider ist es uns noch nicht gelungen ein zufrieden stellendes Ergebnis zu erzeugen. Wir haben aber auch festgestellt, dass der Umgang mit Shader und Blending eine Kunst für sich ist.

4 Projektumgebung

In diesem Abschnitt wollen wir kurz darauf eingehen, mit welchen Mitteln wir gearbeitet haben, wie die Projektstruktur sieht und wie das Programm gestartet wird.

Das Programm wird im VR-Labor am PC an dem der Beamer angeschlossen ist gestartet.

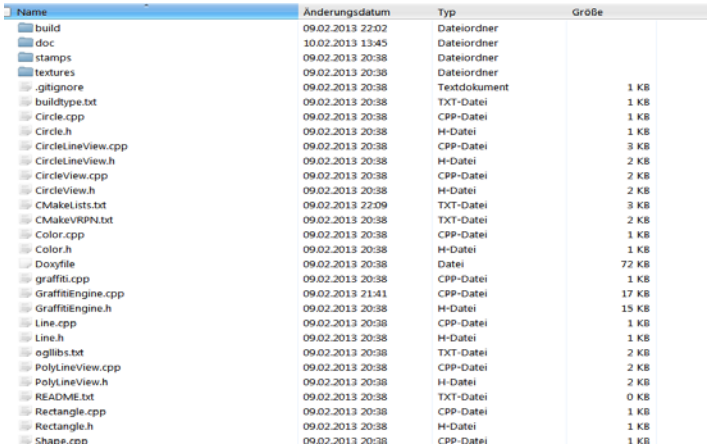
Da auf einem Rechner nicht zwei VRPN Server gleichzeitig laufen können muss zweiter Rechner (in der Regel ist das „davis.ds.fh-kl.de“)als VRPN Server dienen. Auf diesem zweiten Rechner muss die Kinect angeschlossen und der FAAST Server gestartet sein.

Ist die Anwendung gestartet muss zuerst auf der Tastatur ‚p‘ gedrückt werden, dies aktiviert die VRPN Funktion des Programmes. Nun kann mit der Maus und der Tastatur gemalt werden.

Will man mit der WiiMote und dem Tracker malen muss nach dem ‚p‘ gedrückt wurde noch die Taste ‚t‘ gedrückt werden. Dies sagt dem Programm, dass ab jetzt die Tracker Daten verwendet werden sollen.

4.1 Projektstruktur

Die Quellcode Dateien, Doxygen und CMake Dateien liegen alle im selben Verzeichnis, es gibt drei unter Ordner; Textures, Stamps und Doc. Um das Projekt zu bauen wird CMake benutzt. In der Regel wird das Projekt dann in einem Unterordner des Projekts gebaut. Der Ort an dem sich die Texturen befinden ist in einer CMake Datei festgelegt und wird beim „build“ Vorgang in den build Ordner/textures kopiert. Quell Code Dateien hingegen werden nur in den build Ordner verlinkt.



Name	Änderungsdatum	Typ	Größe
build	09.02.2013 22:02	Dateiordner	
doc	10.02.2013 13:45	Dateiordner	
stamps	09.02.2013 20:38	Dateiordner	
textures	09.02.2013 20:38	Dateiordner	
.gitignore	09.02.2013 20:38	Textdokument	1 KB
buildtype.txt	09.02.2013 20:38	TXT-Datei	1 KB
Circle.cpp	09.02.2013 20:38	CPP-Datei	1 KB
Circle.h	09.02.2013 20:38	H-Datei	1 KB
CircleLineView.cpp	09.02.2013 20:38	CPP-Datei	3 KB
CircleLineView.h	09.02.2013 20:38	H-Datei	2 KB
CircleView.cpp	09.02.2013 20:38	CPP-Datei	2 KB
CircleView.h	09.02.2013 20:38	H-Datei	2 KB
CMakeLists.txt	09.02.2013 22:09	TXT-Datei	3 KB
CMakeVRPN.txt	09.02.2013 20:38	TXT-Datei	2 KB
Color.cpp	09.02.2013 20:38	CPP-Datei	1 KB
Color.h	09.02.2013 20:38	H-Datei	1 KB
Doxyfile	09.02.2013 20:38	Datei	72 KB
graffiti.cpp	09.02.2013 20:38	CPP-Datei	1 KB
GraffitiEngine.cpp	09.02.2013 21:41	CPP-Datei	17 KB
GraffitiEngine.h	09.02.2013 20:38	H-Datei	15 KB
Line.cpp	09.02.2013 20:38	CPP-Datei	1 KB
Line.h	09.02.2013 20:38	H-Datei	1 KB
ogllibs.txt	09.02.2013 20:38	TXT-Datei	2 KB
PolyLineView.cpp	09.02.2013 20:38	CPP-Datei	2 KB
PolyLineView.h	09.02.2013 20:38	H-Datei	2 KB
README.txt	09.02.2013 20:38	TXT-Datei	0 KB
Rectangle.cpp	09.02.2013 20:38	CPP-Datei	1 KB
Rectangle.h	09.02.2013 20:38	H-Datei	1 KB
Shape.cpp	09.02.2013 20:38	CPP-Datei	1 KB

4.2 Programme

Verwendet wurden folgende Programme:

CMake 2.8 und höher

Visual Studio 9 2008 Win64

MS Word 2010

IrfanView
Git
FAAST
Doxygen

5 Rückblick und zukünftige Arbeit

5.1 Was würden wir anders machen?

Würden wir die Anwendung erneut erstellen würden wir uns am Anfang mehr Gedanken über die Architektur und Namensregeln machen. Damit ist gemeint Regeln aufstellen die festlegen in welcher Sprache geschrieben wird und wie viel abgekürzt wird.

Wir würden von Anfang an im Code dokumentieren und auch von Anfang an mit einer Versionsverwaltungssoftware wie zum Beispiel Git arbeiten.

5.2 Wie könnte man die Anwendung erweitern oder verbessern?

- Implementierung einer externen Konfigurationsdatei, in der man z.B. einstellen kann in welchem Format die Anwendung laufen soll (4:3, 16:9, ...) und welcher Rechner als VRPN Server dient.
- Verbesserung des Shaders und des Blending, so dass man einen besseren Sprüheffekt erhält.
- Optimierung des Sprühradius; Funktionalität ist zwar implementiert, jedoch sind die Werte nicht realistisch.
- Eine neue Möglichkeit Farben zu wählen; In dem man Sprühdosen in verschiedenen Farben am unteren Rand des Hintergrunds anzeigt. Diese können dann per Geste gewählt werden.
- Mehrere Nutzer
- Sound für Sprühgeräusch und Sound für Bewegung der Sprühdose, damit ist dieses Klicken gemeint.
- Eine Radiergummi-/Wischtuchfunktion
- Das „Handbuch“ der Anwendung weiter ausarbeiten; Anleitung für Gesten
- Texturen mit Alphakanal wählen um den Sprüheffekt dazustellen, das Blending zu erleichtern und so leere (transparente) Pixel zu erlauben die sich überschneiden können um der Herangehensweise von discard entgegenzuwirken. Z. B. mit PNG oder TGA – Formaten arbeiten, um dies zu erreichen.
- Einige Button-Befehle auf Gesten legen, z. B. das heben der linken Hand als Indikator verwenden um Farben rückwärts zu durchlaufen. Dadurch bräuchten wir nur einen Button zur Farbwahl, bzw. für analoge Funktionalitäten die mit Iteration arbeiten.
- Das Erstellen von Formen anhand von Gesten, Form-Button halten und nachzeichnen einer erlaubten Form als Befehl zum Anheften der gewünschten Form am zuvor gewählten Punkt.

6 Literaturverzeichnis

6.1 Dokumente

[1] B. Manfred, „fh-kl.de,“ [Online]. Available: http://www.fh-kl.de/~brill/vr/Assets/downloads/vrpn.pdf.

[2] B. L. S. D. K. a. M. B. E. A. Suma, „FAAST,“ [Online]. Available: http://projects.ict.edu/mxr/faast/.

[3] cplusplus, „cplusplus.com,“ [Online]. Available: http://cplusplus.com/.

[4] Comunity, „Stackoverflow,“ [Online]. Available: http://stackoverflow.com.

[5] M. J. T. L. D. S. S. H. R. R. S. A. Manfred Brill, „VRPN Dokumentation,“ [Online]. Available: http://www.fh-kl.de/~brill/GraphicsEngine/Assets/Docs/index.html.

6.2 Texturen und Logo

http://www.fh-kl.de/~hettel/Logos/fh-kl.gif

http://www.imgbox.de/users/PBForum/BLOG0/mauer_2.jpg

http://www.c4d-jack.de/php/textures/termsofuse.html
http://www.c4d-jack.de/php/textures/gal/SRT_Packages/Part_I/wand_grobstein_1c.jpg
http://www.c4d-jack.de/php/textures/gal/SRT_Packages/Part_II/wand_backstein_2.jpg
http://www.c4d-jack.de/php/textures/gal/SRT_Packages/Part_II/wand_backstein_3.jpg
http://www.c4d-jack.de/php/textures/gal/SRT_Packages/Part_II/boden_asphalt1.jpg

http://freestocktextures.com/texture/id/471

ANMERKUNG:
Dateinamen im Projekt leicht abgeändert da leichter zu schreiben/unterscheiden

wand_grobstein_1c.jpg	->	wand_backstein_1.bpm
wand_backstein_2.jpg	->	wand_backstein_2.bmp
wand_backstein_3.jpg	->	wand_backstein_3.bmp
boden_asphalt1.jpg	->	asphalt_1.bmp
mauer_2.jpg	->	mauer_1.bmp

7 Anhang

7.1 Entwurf eines Klassendiagram für geplante Vererbung