TU WIEN

Stereoframework

Michael Hecher

Inhaltsverzeichnis

Über dieses Dokument....3
Ziel des Frameworks....3
Komponenten....3
Unterstützte Hardware....4
eMagin Z800 3D-Visor HMD....4
DepthQ 120Hz Projektor....4
Zirkulär polarisierte Projektoren....4
nVidia Treiberversion 190 und folgende....4
nVidia Stereo-Treiber....5
Anwendungsbeispiele....5
Beispiel 1: Ein Fenster des Betriebssystems konvertieren....5
Beispiel 2: Erstellen eines Stereofensters....6
GLUT....7
„High Level“ Engines....8
Einführung in das Framework....8
Konstanten....8
Stereo Funktionen....9
Microsoft Windows spezifische Funktionen....10
Matrizen Berechnung....11
Stereo.ini....11
Zu beachten....12
User Interfaces....12
Der Mauszeiger....13
Sky Box....13
OpenGL Funktionen....13
Fensteranwendungen....13
Effekte....13
Stereovideos aufnehmen....13
Programme....13
Qualität der Stereodarstellung....14
Crosstalk (Ghosting)....14
Leuchtkraftverlust....16
Veränderungen des Gamut....16
Referenzen....18

Über dieses Dokument

In diesem Dokument soll über die Verwendung des Stereoframeworks (in Folge kurz Framework genannt) informiert werden. Es ist keine Einführung in Stereorendering, sondern die Erläuterung eines Stereoframeworks. Es wird davon ausgegangen, dass der Leser mit OpenGL und den Grundlagen von Computergrafik vertraut ist.

Ziel des Frameworks

Das Framework dient in erster Linie dazu, die Erstellung von OpenGL-Stereoapplikationen für unterschiedliche Hardware zu vereinfachen und stellt diesbezüglich Funktionen zur Verfügung. Das Framework soll das Portieren von „nicht Stereoapplikationen“ zu Stereoapplikationen einfach gestalten, weshalb darauf geachtet wurde, die Anzahl der Funktionen gering zu halten. Des Weiteren kann das Framework dazu verwendet werden, Stereo-Videos aufzunehmen.

Es handelt sich nicht um ein „High Level Framework“. Es werden keine Funktionen zur Verfügung gestellt, die sich auf das Rendern von Objekten beziehen. Die Implementierung einer Renderengine, die Stereorendering unterstützt, bleibt dem Programmierer überlassen.

Das Framework unterstützt ausnahmslos Render Contexte, welche „Double Buffering“ verwenden, also einen Front- und Back-Buffer besitzen.

Komponenten

Das Framework besteht aus mehreren Komponenten und wurde unter Windows XP und Vista mit den Programmiersprachen C/C++ und Pascal (Delphi) getestet. Es werden, je nach Programmiersprache und Applikation, nicht alle Teile zum Erstellen einer stereofähigen Applikation benötigt.

Stereo.h

C/C++ Header mit Definitionen von Objekten und Funktionen. Diese Datei wird benötigt, um einer Applikation die Verwendung des Stereoframeworks in Sprachen wie C und C++ zu ermöglichen. Die Headerdatei befindet sich im *include* Verzeichnis des Frameworks.

Stereo.cpp

In dieser Datei sind die Funktionen, welche in der Header Datei angeführt sind, implementiert. Sie kann von in C und C++ programmierten Programmen in ein Projekt direkt eingebunden werden, um das Debuggen des Frameworks zu vereinfachen. Sie ist auch als Ressource gedacht, um die Prinzipien von Stereorendering zu verstehen. Die Datei befindet sich im *src* Verzeichnis des Frameworks.

Stereo.pas

Pascal Code, um das Einbinden der Stereo.dll in Pascal basierten Projekten zu ermöglichen. Die Datei befindet sich im *include* Verzeichnis des Frameworks.

Stereo.lib

Muss von Visual C und C++ Projekten eingebunden werden, wenn das automatische Laden der DLL-Funktionen ermöglicht werden soll. Die Dateien befinden sich im *lib* Verzeichnis des Frameworks.

Stereo.dll

Dynamische Bibliothek, die in unterschiedlichen Programmiersprachen unter Windows

Betriebssystemen verwendet werden kann. Die DLL muss sich in einem geeigneten Verzeichnis befinden (z.B. im gleichen Verzeichnis wie die Applikation), wenn die Funktionen der DLL automatisch geladen werden soll. Die Dateien befinden sich im *dll* Verzeichnis des Frameworks.

Unterstützte Hardware

Es gibt unterschiedliche Hardware, um Stereodarstellungen zu ermöglichen. Dabei kann es sich um Monitore, Projektoren oder Head Mounted Displays (HMD) handeln. In diesem Abschnitt wird jene Hardware erklärt, die durch das Framework unterstützt wird.

eMagin Z800 3D-Visor HMD

Um Stereorendering über das Z800 darzustellen, muss auf diesem die Firmware 6.38 oder neuer installiert sein. Ist dies nicht der Fall, kann das Stereorendering API von eMagin nicht verwendet werden. Das Z800, mit älterer Firmware funktioniert mit den Stereo-Treibern von nVidia (siehe [8]).

DepthQ 120Hz Projektor

Dieser Projektor projiziert Bilder mit einer Frequenz von 120 Hz und wird mit Shutterglasses kombiniert. Dabei wird abwechselnd ein Bild für das linke und das rechte Auge abgebildet. Die Shutterglasses verhindern, dass das Bild für das linke Auge für das rechte sichtbar ist und umgekehrt, indem sie mit der gleichen Frequenz abwechselnd das Sichtfeld der Augen verdecken.

Das Framework verwendet den OpenGL Quad Buffer, um Stereoszenen über den DepthQ Projektor darzustellen. Im Gegensatz zum „normalen" OpenGL Render Buffer kann beim Quad Buffer zwischen einem linken und einem rechten Buffer umgeschaltet werden [9].

Zirkulär polarisierte Projektoren

Mit zwei unterschiedlich polarisierten Projektoren werden zwei Bilder gleichzeitig auf eine silberbeschichtete Leinwand projiziert. Ein Projektor erzeugt das Bild für das linke und der andere für das rechte Auge. Sie werden wie ein Multi-Monitor-System an den Computer angeschlossen. Es kann für die Stereodarstellung der OpenGL Quad Buffer (wenn von der Grafikkarte und dem Treiber unterstützt) oder ein vergrößerter OpenGL Render Context (zweifache Breite für linken und rechten Monitor wie in Abbildung 1) benutzt werden. Für letzteren Fall wird mit der glViewport Funktion, zwischen linkem und rechtem Monitor umgeschaltet. Neben einem vergrößerten OpenGL Render Context besteht weiters die Möglichkeit, einen Render Context zwischen zwei Fenstern zu teilen und mit der *wglMakeCurrent* Funktion zwischen diesen „umzuschalten". Bei Tests führte dies zu Performanceeinbußen von über 50% gegenüber der glViewport-Lösung.

glViewport(0, 0, width/2, height)	glViewport(width/2, 0, width/2, height)

Abbildung 1: Verwendung von glViewport, um eine Stereoszene über zwei polarisierte Projektoren darzustellen.

nVidia Treiberversion 190 und folgende

Mit den *GeForce*-Treibern ab Version 190, wurde der Stereotreiber in die „normalen" *GeForce*-Treiber integriert. Stereodarstellungen sind somit auch für nVidia Grafikkarten der *GeForce* Reihe

möglich. Der OpenGL Quad Buffer ist nicht mehr Quadro Karten vorbehalten. Die neuen Treiberversionen ermöglichen es, den Grafiktreiber, für bestimmte Hardware einzurichten. Welche Karten und Hardware *nVidia Stereo Vision* unterstützen, kann auf der nVidia Webseite nachgelesen werden [10].

nVidia Stereo-Treiber
Die Stereo-Treiber von nVidia werden durch das Framework nicht direkt angesprochen. Sie schalten sich automatisch in die Render Pipeline ein und berechnen anhand des Depth-Buffers/Z-Buffers das zweite Bild [7]. Das Framework darf nicht, wie in den anderen Fällen, die Szene für beide Augen darstellen, da dies von den Treibern übernommen wird. Es wird in diesem Falle sichergestellt, dass kein Stereorendering durch das Framework auftritt.

Anwendungsbeispiele

Es bestehen verschiedene Anwendungsmöglichkeiten für das Framework, welche durch Applikation und Anwendungsfall bestimmt sind. Applikationen lassen sich dahingehend unterscheiden, ob sie auf „high level“ 3D Engines und Framewoks aufsetzen, welche bereits Stereorendering unterstützen, oder nicht. Anwendungsfälle sind das Portieren (siehe „Beispiel 1“) oder Erstellen (siehe „Beispiel 2“) einer stereofähigen Applikation.

Für die verschiedenen Anwendungsmöglichkeiten werden in diesem Abschnitt Beispiele angeführt, die die Verwendung des Frameworks erläutern.

Beispiel 1: Ein Fenster des Betriebssystems konvertieren.
Um Stereorendering in einer Applikation für unterschiedliche Hardware zu ermöglichen, muss zuerst das Fenster mittels der *stereoConvertWindow* Funktion in das in der *stereo.ini* Datei angegebene Format konvertiert werden. In der draw-Funktion wird die Szene für ein Auge gerendert und das Stereo Target getauscht. Im nächsten Frame wird das zweite Auge gerendert und das Stereo Target wieder getauscht.

```
// main.cpp
int MAIN(...)
{
…
   StereoOptions stereoOptions;

   // lade die standard Optionen aus der stereo.ini Datei
   stereoLoad(&stereoOptions);

   // Konvertiere ein Fenster, welches das in stereo.ini festgelegte
   // Format besitzt. Die Auflösung des Fensters soll 800x600 Pixel sein.
   stereoConvertWindow(&stereoOptions, hWnd, 800, 600);
…
}

// drawFunktion.cpp
void draw(void)
{
   // vertausche das Stereo Target (linkes / rechtes Auge)
   stereoSwapStereoTarget();

   // lösche Farb- und Tiefeninformationen (wenn erlaubt)
   stereoGlClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

   // setze OpenGL PERSPECTIVE matrix
```

```
    stereoGluPerspectiveRHCM(camFOV, screenW/screenH, camNear, camFar);

    // setze OpenGL MODEL_VIEW matrix
    stereoGluLookAtRHRM(camPos[0], camPos[1], camPos[2],
        camLookAt[0], camLookAt[1], camLookAt[2],
        camUp[0], camUp[1], camUp[2]);

    //// berechne PERSPECTIVE und MODEL_VIEW Matrizen
    //float perspective[16];
    //float view[16];
    //stereoGluPerspectiveRHCM(camFOV, screenW/screenH, camNear, camFar,
    //    (float*)perspective);
    //stereoGluLookAtRHRM(camPos[0], camPos[1], camPos[2],
    //    camLookAt[0], camLookAt[1], camLookAt[2],
    //    camUp[0], camUp[1], camUp[2], (float*)view);

    // zeichne die Szene
    drawScene();

    // rufe SwapBuffers auf (wenn erlaubt)
    stereoGlPresent();
}
```

Code 1: Ein bereits bestehendes Fenster konvertieren. (Programiersprache C)

Schritt 1: Laden der Stereoeinstellungen

Im ersten Schritt müssen die Einstellungen der *stereo.ini* Datei geladen werden. Hierfür wird die Funktion *stereoLoad* verwendet, welche einen Zeiger auf eine *StereoOptions* Struktur erwartet. Es können die einzelnen Member dieser Struktur auch im Programmcode gesetzt werden. Das Programm muss dann gegebenenfalls für unterschiedliche Hardware neu kompiliert werden.

Schritt 2: Konvertieren oder Erstellen des Fensters

Mit den geladenen Stereooptionen kann ein neues Fenster erzeugt (*stereoCreateWindow* in Verbindung mit *stereoCreateOpenGL*, siehe „Beispiel 2“) oder ein bereits bestehendes Fenster, wie im obigen Beispiel, mit der Funktion *stereoConvertWindow* konvertiert werden.

Schritt 3: Laden und/oder Berechnen der Matrizen

Bevor die Objekte gerendert werden können, müssen noch die Projektions- und die Kameramatrix für das jeweilige Auge berechnet werden. Dies kann über die Funktionen *stereoGluPerspective[RH|LH][CM|RM]* und *stereoGluLookAt[RH|LH][CM|RM]* geschehen oder über „eigene“ Berechnungsfunktionen. Wichtig ist beim Verwenden der vom Framework zur Verfügung gestellten Funktionen, dass das aktuelle Stereotarget mit *stereoSwapStereoTarget* vertauscht wird, um die Matrizen für beide Augen zu berechnen.

Schritt 4: SwapBuffers

Nach dem Rendern der Szene muss die SwapBuffers-Funktion aufgerufen werden, um die Szene zu präsentieren. Es gibt Ausnahmen (zum Beispiel bei der Darstellung von zwei Viewports), in denen das Aufrufen der SwapBuffers-Funktion zu einer fehlerhaften Darstellung führt, weshalb vom Framework die *stereoGlPresent* Funktion zur Verfügung gestellt wird. Diese Funktion ruft SwapBuffers nach den eingestellten Stereooptionen auf, um Darstellungsfehler zu verhindern.

Beispiel 2: Erstellen eines Stereofensters.

Wenn beim Portieren eines bestehenden Programms keine Möglichkeit besteht, Informationen über das Fensters zu erhalten, und daher ein neues Fenster für eine Stereodarstellung erzeugt werden

muss, kann mit Hilfe des Frameworks ein Fenster und/oder ein OpenGL Render Context erzeugt werden. Ein Beispiel sind Fensteranwendungen mit Menüleiste und anderer GUI-Elementen.

```
// main.cpp

// Callback Funktion zum Behandeln von Nachrichten.
LRESULT CALLBACK MessageHandler(
   HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
…
}

int MAIN(..., HINSTANCE hInstance, ...)
{
…
   StereoOptions stereoOptions;

   // lade die standard Optionen aus der stereo.ini Datei
   stereoLoad(&stereoOptions);

   // Erzeuge ein Fenster, welches das in stereo.ini festgelegte
   // Format besitzt. Die Auflösung des Fensters soll 800x600 Pixel sein.
   stereoCreateWindow(&stereoOptions,   hInstance,   TEXT("WndClassName"),
                      MessageHandler, 800, 600);
   // Optional: Erzeuge OpenGL Render Context.
   stereoCreateOpenGL();

// Main Loop, etc.
…

   // Das Fenster und den Render Context zerstören.
   // Der Render Context muss nur zerstört werden, wenn er auch erzeugt
   // wurde.
   stereoDestroyOpenGL();
   stereoDestroyWindow();
}

// drawFunktion.cpp
void draw(void)
{
…
}
```

Code 2: Ein Fenster mit dem Stereoframework erzeugen.

Der Unterschied zum vorhergehenden Beispiel besteht darin, dass *stereoCreateWindow* zum Erzeugen eines Fensters und *stereoCreateOpenGL* zum Erzeugen eines Render Contextes verwendet wurden (siehe „Einführung in das Framework“ für Details zu diesen Funktionen). Vor dem Beenden des Programms können das Fenster und der Render Context mit den Funktionen *stereoDestroyWindow* bzw. *stereoDestroyOpenGL* zerstört werden.

GLUT

Für Programme die GLUT verwenden ist es nicht ausreichend, über die *stereoConvertWindow* Funktion des Frameworks, das Fenster zu konvertieren. Um in diesem Fall den OpenGL Quad Buffer zu aktivieren, muss der GLUT-Funktion *glutInitDisplayMode GLUT_STEREO* übergeben werden. Alle anderen durch das Framework unterstützten Rendermodi, werden weiterhin über *stereoConvertWindow* realisiert.

```
// main.cpp

int MAIN(...)
{
…

   StereoOptions stereoOptions;
   // lade Optionen aus der stereo.ini Datei
   stereoLoad(&stereoOptions);

…

   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH |
       ((stereoOptions.mode&stereoQuadBuffer)?GLUT_STEREO:0));
…

   stereoConvertWindow(&stereoOptions, 0, winWidth, winheight);

…
}
```
Code 3: GLUT Beispielanwendung.

„High Level“ Engines

Manche 3D Engines unterstützen Stereorendering und müssen daher nicht portiert werden. Für diesen Fall und für den Fall, dass eine Engine nicht mit dem Framework angepasst werden kann, muss der Engine über die *StereoOptions* Struktur der Stereomodus mitgeteilt werden. Hierzu reicht es, die *stereoLoad* Funktion wie in den obigen Beispielen aufzurufen und die geladenen Informationen in der *StereoOptions* Struktur an die Engine weiterzureichen.

Einführung in das Framework

Konstanten

`STEREORENDERMODE`

Konstanten vom Typ *STEREORENDERMODE* geben an, welche Stereorenderingtechnik verwendet wird.

`stereoDepthQ / stereoQuadBuffer`

Diese Technik kann verwendet werden, um eine Szene mit dem unter „DepthQ 120Hz Projektor“ beschriebenen DepthQ Projektor darstellen zu können.

`stereoHMD_Z800`

Diese Technik kann verwendet werden, um eine Szene mit dem unter „eMagin Z800 3D-Visor HMD“ beschriebenen HDM von eMagin darstellen zu können.

`stereoTwoViewports`

Diese Technik kann verwendet werden, um eine Szene mit den unter „Zirkulär polarisierte Projektoren“ beschriebenen Projektoren darstellen zu können. Diese Einstellung kann auch benutzt werden, um Stereovideos für eben erwähnte Projektoren aufzunehmen.

`stereoWindowed`

Diese Option teilt dem Framework mit, dass die Szene in einem Fenster dargestellt werden soll. Sie ist für Debuggingzwecke vorgesehen und kann mit *stereoTwoViewports*, *stereoDepthQ* oder *stereoHMD_Z800* durch „|“ (logisches oder) verknüpft werden. Wird *stereoWindowed* nicht angeführt, wird die Szene im Vollbildmodus gerendert.

`stereoWindowedNoBorder`
Wie *stereoWindowed*. Das Fenster hat jedoch keine Titelleiste. Dieser Modus ist speziell für das Aufnehmen von Stereovideos vorgesehen. Das Fenster wird in der linken oberen Ecke des Desktops platziert, damit der Aufnahmebereich für das Video einfacher festlegt werden kann.

`STEREOTARGET`
Konstanten vom Typ *STEREOTARGET* beschreiben Render Targets. Render Targets sind die Ziele des Rendervorgangs. Für das linke und rechte Auge gibt es unterschiedliche Render Targets, da für jedes Auge ein anderes Bild gerendert werden muss.

`monoCenter`
Ein Render Target für Monorendering (normales rendern ohne Stereoeffekt). Es gibt nur ein Target (Center).

`stereoLeft/stereoRight`
Ein Render Target für Stereorendering. Für Stereorendering gibt es zwei Targets (left/right) für das linke und das rechte Auge.

Stereo Funktionen
`bool stereoEnabled(void)`
Gibt *true* zurück, wenn Stereorendering möglich ist. Die Voraussetzung hierfür ist die korrekte Initialisierung des Frameworks.

`void stereoGlPresent(void)`
Ruft die Funktion *SwapBuffers* auf, sofern dies für den aktuellen Rendermodus notwendig ist.

`void stereoGlClear(GLbitfield)`
Je nach Rendermodus kann es sein, dass bestimmte Bits von *Glbitfield* nicht gesetzt werden dürfen. Diese Funktion ruft *glClear* auf und entfernt diese Bits.

`bool stereoLoad(StereoOptions* options)`
Lädt eine *StereoOptions* Struktur aus einer *stereo.ini* Datei, welche sich im selben Ordner wie die Anwendung oder im „Windows"-Ordner befinden muss.

`bool stereoValid(void)`
Gibt *true* zurück, wenn das Framework richtig und vollständig initialisiert wurde. Eine korrekte Initialisierung des Frameworks ist Voraussetzung für Stereorendering. Ob Stereorendering zu Verfügung steht, muss mit der Funktion *stereoEnabled* geprüft werden.

`void stereoSwapStereoTarget(void)`
Tauscht das zu zeichnende Auge (Target). Wird Stereorendering nicht verwendet, hat diese Funktion keine Auswirkungen.

`void stereoSetEyeSeparation(float)`
Legt die zu verwendende Distanz zwischen linkem und rechtem Auge fest.

`float stereoGetEyeSeparation(void)`
Gibt die verwendete Distanz zwischen linkem und rechtem Auge zurück.

`void stereoSetEyeSwap(float)`
Legt fest, ob linkes und rechtes Auge vertauscht werden sollen. Ein Wert von -1 vertauscht linkes und rechtes Auge. Ein Wert von 1 stellt den ursprünglichen Zustand wieder her.

`float stereoGetEyeSwap(void)`
Gibt 1 oder -1 zurück, je nachdem ob linkes und rechtes Auge vertauscht wurden oder nicht. Bei einem Wert von -1 wurden die Augen vertauscht.

```
void stereoSetFocalLength(float)
```
Legt die Entfernung zwischen Linse und Brennpunkt fest (Brennweite).

```
float stereoGetFocalLength(void)
```
Gibt die verwendete Entfernung des Brennpunktes zur Linse zurück (Brennweite).

```
bool stereoSetRenderMode(STEREORENDERMODE mode)
```
Ermöglicht es, den Rendermodus zu ändern. Siehe auch STEREORENDERMODE.

```
STEREORENDERMODE stereoGetRenderMode(void)
```
Gibt den momentan verwendeten Rendermodus zurück.

```
bool stereoSetStereoTarget(STEREOTARGET target)
```
Setzt das zu verwendende „stereo target“ (linkes Auge, rechtes Auge, mono).

```
STEREOTARGET stereoGetStereoTarget(void)
```
Gibt das momentan verwendete „stereo target“ zurück (linkes Auge, rechtes Auge, mono).

Microsoft Windows spezifische Funktionen

```
bool stereoCreateWindow(
      StereoOptions* options,
      HINSTANCE hInstance,
      WCHAR* wndClassName,
      WNDPROC MessageHandler,
      int width,
      int height)
```
Erzeugt ein Fenster mit den angegebenen Stereooptionen. Die *StereoOptions* Struktur kann mit Hilfe der *stereoLoad* Funktion geladen werden.

```
bool stereoConvertWindow(
      StereoOptions* options,
      HWND window = 0,
      int width = 0,
      int height = 0)
```
Konvertiert ein bestehendes Fenster in das in *options* angegebene Format. *window* ist der Handle des zu konvertierenden Fensters. *width* und *height* geben die gewünschte Breite und Höhe an. Werte von 0 übernehmen die Breite und Höhe des zu konvertierenden Fensters. Die Funktion erstellt keinen neuen OpenGL Render Context, wodurch ein Wechseln in den Quad Buffer Mudus nicht möglich ist (siehe Abschnitt „Anwendungsbeispiele“).

```
void stereoDestroyWindow(void)
```
Zerstört das über *stereoCreateWindow*, *stereoConvertWindow* oder *stereoSetWindow* erstellte bzw. festgelegte Fenster.

```
bool stereoCreateOpenGL(
      int color = 32,
      int depth = 32,
      int stencil = 0,
      int accum = 0,
      int aux = 0,
      int multisampling = 0,
      int openglVersion = 2)
```
Erzeugt einen Render und Device Context für das über *stereoCreateWindow*, *stereoConvertWindow* oder *stereoSetWindow* erstellte bzw. festgelegte Fenster.

```
void stereoDestroyOpenGL(void)
```
Zerstört den über *stereoCreateOpenGL* oder *stereoGlSetRenderContext* erstellten bzw. festgelegten Render Context.

`void stereoGlSetDeviceContext(HDC deviceContext)`
Diese Methode setzt den momentanen Device Context. Das Framework stellt die Funktion *stereoGlCreateStereoWindow* bereit, um ein Fenster mit Device Context zu erzeugen. Sollte der Context bereits existieren, kann er mit dieser Funktion übernommen werden.

`HDC stereoGlGetDeviceContext(void)`
Diese Funktion gibt den momentan verwendeten Device Context zurück.

`void stereoGlSetRenderContext(HGLRC renderContext)`
Diese Methode setzt den momentanen OpenGL Render Context. Das Framework stellt die Funktion *stereoGlCreateStereoWindow* bereit, um ein Fenster mit OpenGL Render Context zu erzeugen. Sollte ein Context bereits existieren, kann er mit dieser Funktion übernommen werden.

`HGLRC stereoGlGetRenderContext(void)`
Diese Funktion gibt den momentan verwendeten OpenGL Render Context zurück.

`void stereoSetWindow(HWND wnd)`
Diese Funktion setzt das zu verwendende Fenster. Diese Funktion kann verwendet werden, um ein bereits erstelltes Fenster zu übernehmen.

`HWND stereoGetWindow(void)`
Diese Funktion gibt den Handle des momentan verwendeten Fensters zurück.

Matrizen Berechnung
Es stehen Hilfsfunktionen für verschiedene Koordinatensysteme und Darstellungsstrategien zur Verfügung, welche anhand der Endung einer Funktion identifiziert werden können. So steht LH für „left handed“ bzw. RH für „right handed“ Koordinatensystem. Wird eine Funktion mit der Endung CM verwendet, liegt die resultierende Matrix in „column major“ Form und bei RM in „row major“ Form vor.

```
void stereoGluLookAt[RH|LH][RM|CM](
      GLfloat eyex, GLfloat eyey, GLfloat eyez,
      GLfloat centerx, GLfloat centery, GLfloat centerz,
      GLfloat upx, GLfloat upy, GLfloat upz,
      float* matrix = 0);
```

Berechnet die Model-View-Matrix. Die übergebenen Werte sind äquivalent mit jenen der OpenGL *gluLookAt* Funktion. Wird der Funktion für den Parameter *matrix* NULL übergeben, so wird das Ergebnis in die aktuelle OpenGL Matrix geladen. Zeigt der Zeiger auf ein Array, welches mindestens 16 Elemente fassen muss, so wird das Ergebnis in das angegebene Array geschrieben.

```
void stereoGluPerspective[RH|LH][RM|CM](
      GLfloat fovy,
      GLfloat aspect,
      GLfloat near,
      GLfloat far,
      float* matrix = 0);
```

Berechnet die Projektionsmatrix. Die übergebenen Werte sind äquivalent mit jenen der OpenGL *gluPerspective* Funktion. Wird der Funktion für den Parameter *matrix* NULL übergeben, so wird das Ergebnis in die aktuelle OpenGL Matrix geladen. Zeigt der Zeiger auf ein Array, welches mindestens 16 Elemente fassen muss, so wird das Ergebnis in das angegebene Array geschrieben.

Stereo.ini

Die *stereo.ini* Datei enthält Informationen darüber, welcher Stereomodus und welche Fenster- bzw. Stereoeinstellungen initialisiert werden soll.

Ein Beispiel für eine gültige *stereo.ini* Datei ist:

```
[STEREO]
RenderMode=split
WindowMode=windowed
EyeSeperation=0.6
FocalLength=20
SwapEyes=false
```

Code 4: Inhalt einer gültigen stereo.ini Datei.

`[STEREO]`

Hier beginnen die Einstellungen für das Stereoframework.

`RenderMode=[none|split|quadbuffer|depthq|z800]`
Legt den Rendermodus fest.

none ..... kein Stereorendering
split ..... zwei Viewports verwenden
quadbuffer / depthq ... den Quadbuffer verwenden
z800 ..... das eMagin Z800 initialisieren (ab Firmware 6.38)

`WindowMode=[fullscreen|windowed|windowednoborder]`
Legt fest, ob das Fenster eine Titelleiste hat oder im Vollbildmodus ausgeführt wird.

Fullscreen..... Vollbildmodus
Windowed..... Fenster mit Titelleiste
Windowednoborder .... Fenster ohne Titelleiste

`EyeSeperation=0.6`
Legt die Standard Distanz zwischen linkem und rechtem Auge fest (Einheit Meter).

`FocalLength=20`
Legt den Abstand zwischen Linse und Brennpunkt (Brennweite) fest.

`SwapEyes=false`
Gib an, ob das linke und rechte Auge vertauscht werden sollen oder nicht.

Zu beachten

User Interfaces
Beim Rendern von User Interfaces sind Stereoeffekte zumeist nicht gewollt und müssen unterdrückt werden. Bei nVidias Stereotreibern ist zu beachten, dass die homogenen Koordinaten (w-Koordinate) der Pixel im Framebuffer für das Berechnen der Stereobilder verwendet werden [7]. Daher müssen die w-Koordinate für UI-Elemente, die mit der Grafik API gerendert werden, auf 1.0 gesetzt werden.

Bei der Darstellung über zwei Viewports müssen sich die gleichen UI-Elemente auf jedem Monitor an derselben Stelle befinden. Andernfalls wären sie für ein Auge nicht sichtbar. Hier ist es am einfachsten ein neues stereofähiges Fenster ohne UI-Elementen im Vollbildmodus zu erzeugen, in welchem die Szene in Stereo betrachtet werden kann. Wird Stereorendering nicht mehr benötigt, kann das Fenster geschlossen und mit dem User Interface weitergearbeitet werden. Mit dem OpenGL Quad Buffer oder dem Z800 von eMagin kann auch in Fensteranwendungen Stereorendering verwendet werden.

Der Mauszeiger
Der Standardmauszeiger wird nicht als 3D-Objekt gerendert und daher in manchen Rendereinstellungen falsch dargestellt. Beispielsweise ist der Mauszeiger bei der zwei Viewport Darstellung nur für ein Auge sichtbar, da er auf einem einzigen Monitor dargestellt wird. Eine mögliche Lösung besteht darin, den Standardmauszeiger zu verbergen und einen eigenen Mauszeiger für beide Viewports als Billboard zu zeichnen (mit w-Koordinate 1.0).

Des Weiteren ist zu beachten, dass sich manche Objekte aufgrund des Stereoeffektes vor dem Mauszeiger befinden. Wird der Mauszeiger im 3D Raum nicht vor das Objekt gerückt oder vom Objekt verdeckt, hat dies eine negative Auswirkung auf die Glaubwürdigkeit der Stereodarstellung.

Sky Box
Für Sky Boxes, die am Anfang der Szene gerendert werden (mit z-write deaktiviert), sollte die w-Koordinate größer sein als jene von Umgebungsobjekten [7]. Andernfalls ist der Stereoeffekt des Himmels, wenn der nVidia Treiber für die Stereodarstellung verwendet wird, stärker als jener der Umgebung, obwohl der Himmel am weitesten von der Kamera entfernt ist.

OpenGL Funktionen
Da sich bei manchen Rendermodi der Ursprung des Fensters von (0,0) nach (width/2,0) verschiebt, liefern einige Funktionen andere Werte als erwartet. Beispiele hierfür sind z. B. *glViewport*, *glRasterPos2iv*, *glReadPixels*, *glScissor*, usw. Der verschobene Ursprung muss bei diesen Funktionen berücksichtigt werden.

Fensteranwendungen
Fensteranwendungen eignen sich nicht für Stereorendering, wenn dies von der Hardware nicht unterstützt wird. Es muss der OpenGL-Teil des Fensters auch auf dem zweiten Monitor an selbiger Stelle dargestellt werden (siehe „User Interfaces“). Eine Möglichkeit, dies zu umgehen besteht darin, das User Interface ebenfalls über OpenGL zu rendern und die Applikation im Vollbildmodus auszuführen.

Effekte
Effekte, wie sie in Computerspielen Anwendung finden, können in Stereoapplikationen zu Problemen und dem Verlust der Glaubwürdigkeit der Stereodarstellung führen. Das Paper „Introduction to Stereo Rendering“ [1] geht auf einige Effekte, wie Reflektionen, Parallax Mapping, Bloom, Depth of Field, HDR Rendering oder Fur, in Bezug auf Stereorendering ein und erläutert diese hinsichtlich ihrer Stereoverträglichkeit.

Stereovideos aufnehmen

Programme
Für die Aufnahme von Stereovideos eignet sich jedes Screencapturing Programm wie *FRAPS* [6] oder *CamStudio* [5]. Das Aufnehmen von mehreren Bildschirmen wird von diesen Programmen nicht unterstützt. Das Programm *My Screen Recorder* [4] erlaubt es auch, den Aufnahmebereich über mehrere Monitore zu strecken. Es besteht die Möglichkeit, Videos über bestimmte Frameworkeinstellungen aufzunehmen. In der *Stereo.ini* Datei kann für die Fensterdarstellung *windowednoborder* und für den Stereomodus die Option „split“ (zwei Viewports) gewählt werden, um das Programm als Fenster ohne Rahmen darzustellen. Linkes und rechtes Auge werden nebeneinander dargestellt. Mit dem Aufnahmeprogramm kann nun die aufzunehmende Region an das Fenster angepasst und die Aufnahme gestartet werden.

Da die meisten angeführten Programme eine Aufnahme von mehreren Monitoren nicht unterstützten, ist die Auflösung des Videos durch die Bildschirmauflösung begrenzt. Beispielsweise darf ein Programm im Rendermodus „split“ für einen Bildschirm mit der Auflösung 1980x1080 maximal eine Breite von 1980/2 Pixeln haben, da durch die Option „split“ die Breite des Fensters verdoppelt wird.

Das Abspielen von Videos ist mit dem *Stereoscopic Video Player* von nVidia [3] oder mit dem *StereoMovie Player* [2] möglich.

Qualität der Stereodarstellung

Crosstalk (Ghosting)
Bei mancher Hardware kann es zu störenden Effekten bei der Stereodarstellung kommen. Einer dieser Effekte ist Crosstalk (auch Ghosting genannt). Crosstalk tritt bei Hardware auf, die für linkes und rechtes Auge die gleiche physische Bildquelle verwendet, wie es beispielsweise bei 120Hz oder zirkulär polarisierten Projektoren der Fall ist. Wird die Projektionsfläche direkt betrachtet, sind beide Bilder für jedes Augen sichtbar. Um den Stereoeffekt zu erzielen werden Vorrichtungen wie Shutterglasses oder Polarisationsfilter verwendet, welche die Bilder für linkes und rechtes Auge trennen.

Kommt es bei der Trennung zu Fehlern, ist für das linke Auge ein Teil des rechten Bildes sichtbar und umgekehrt. In diesem Fall spricht man von Crosstalk oder Ghosting.

Dabei ist zwischen Viewer- und System-Crosstalk zu unterscheiden. Während der System-Crosstalk ein Qualitätsfaktor für Vorrichtungen (Systems) wie Shutterglasses ist, stellt der Viewer-Crosstalk einen Qualitätsfaktor für das Empfinden des Benutzers für die bestimmten Kontrastverhältnisse auf der Projektionsfläche dar. Formeln für die Berechnung dieser Größen beschreiben Kuo-Chung Huang et al. in [11] (siehe Formel 1).

$$system\ crosstalk = \frac{\beta_2}{\alpha_2}$$

$$viewer\ crosstalk = \frac{B\alpha_2}{A\alpha_1}\frac{\beta_2}{\alpha_2}$$

A ..... Die Leuchtstärke des linken Bildes für das linke Auge.
B ..... Die Leuchtstärke des rechten Bildes für das rechte Auge.
α_1, α_2 .... Relative Leuchtstärke, die an das richtige Auge übertragen wird.
β_1, β_2 .... Relative Leuchtstärke, die an das falsche Auge übertragen wird.

Formel 1: Die Formel zur Berechnung des Viewer-Crosstalk.

Die Stärke des Crosstalk ist, nach Formel 1, abhängig vom Kontrast $\left(\frac{B\alpha_2}{A\alpha_1}\right)$ zwischen den Bildpunkten des linken und rechten Bildes, die sich an der gleichen Stelle der Projektionsfläche befinden [11] (siehe Abbildung 2).

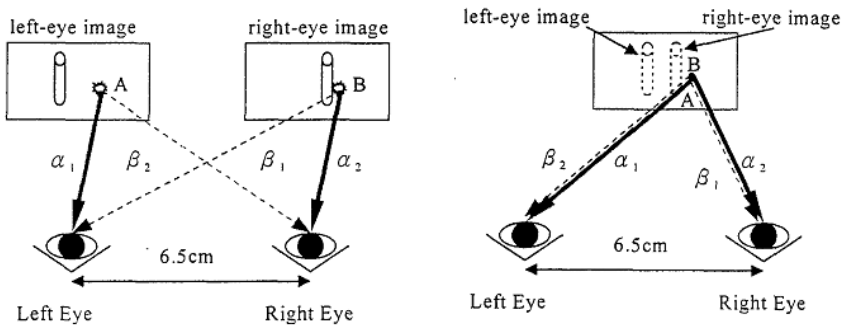


Abbildung 2: Veranschaulichung von Formel 1. (Quelle [11])

Zusätzlich ist der Blickwinkel von Bedeutung. Je höher der Kontrast oder der Blickwinkel, desto höher der Crosstalk (siehe Abbildung 3 und Abbildung 4).

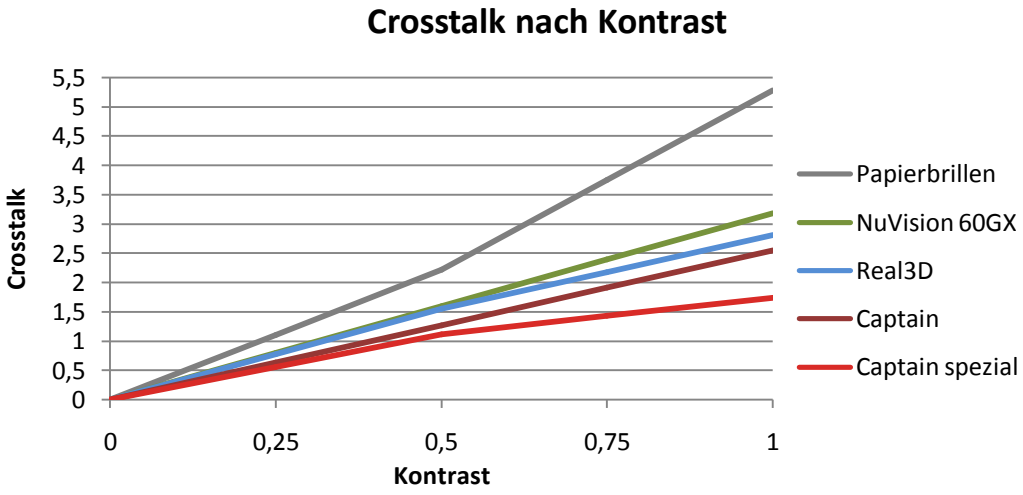


Abbildung 3: System-Crosstalk in Abhängigkeit vom Kontrast. Shutterglasses: NuVision 60GX. Polarisationsfilter: Papierbrillen, Real3D, Captain , Captain Spezialanfertigung.

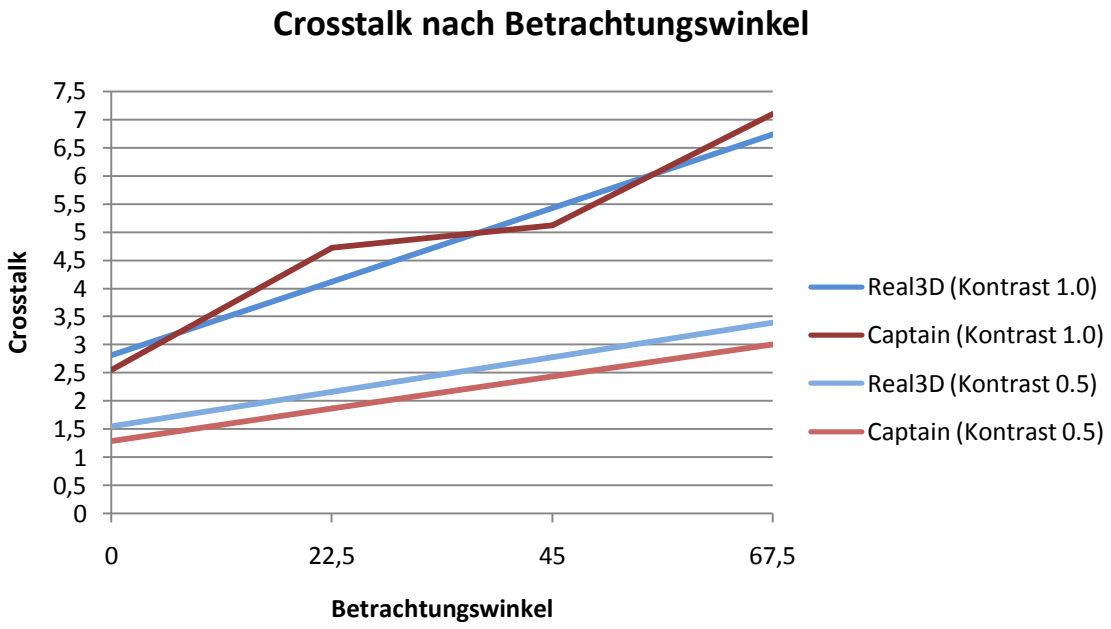


Abbildung 4: System-Crosstalk in Abhängigkeit vom Blickwinkel und unterschiedlichen Polarisationsfiltern für zirkulär polarisierte Projektoren. Ein Blickwinkel von 0° entspricht einem Betrachter, der im rechten Winkel die Bildquelle betrachtet.

Leuchtkraftverlust

Wird eine Projektionsfläche betrachtet, auf welcher die Bilder für das linke und rechte Auge dargestellt werden, besitzt diese, für die Position, an der sich die Augen des Betrachters befinden, eine bestimmte Leuchtkraft. Shutterglasses, Polarisationsfilter und andere Vorrichtungen führen zum Verlust von Leuchtkraft. Dieser kann als Qualitätsfaktor für die Stereohardware angesehen werden. Dabei ist es wünschenswert, dass der Leuchtkraftverlust möglichst gering ausfällt. In Abbildung 5 werden zwei Polarisationsfilter und eine Shutterglasses Vorrichtung in Bezug auf ihren Leuchtkraftverlust verglichen.

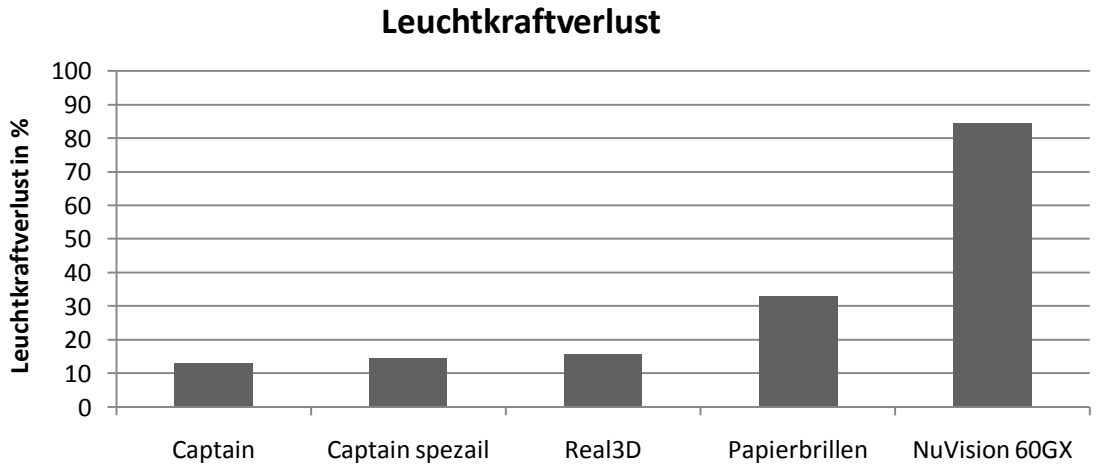


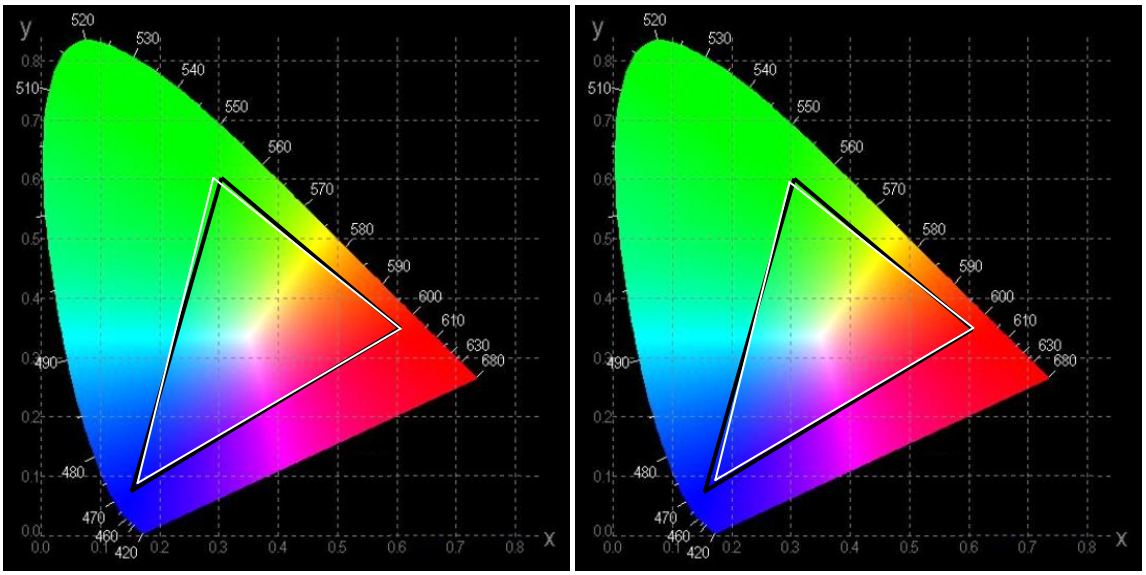
Abbildung 5: Veränderung der Leuchtkraft durch Polarisationsfilter und Shutterglasses in Prozent. Polarisationsfilter: Captain, Captain Spezialanfertigung, Real3D, Papierbrillen. Shutterglasses: NuVision 60GX.

Veränderungen des Gamut

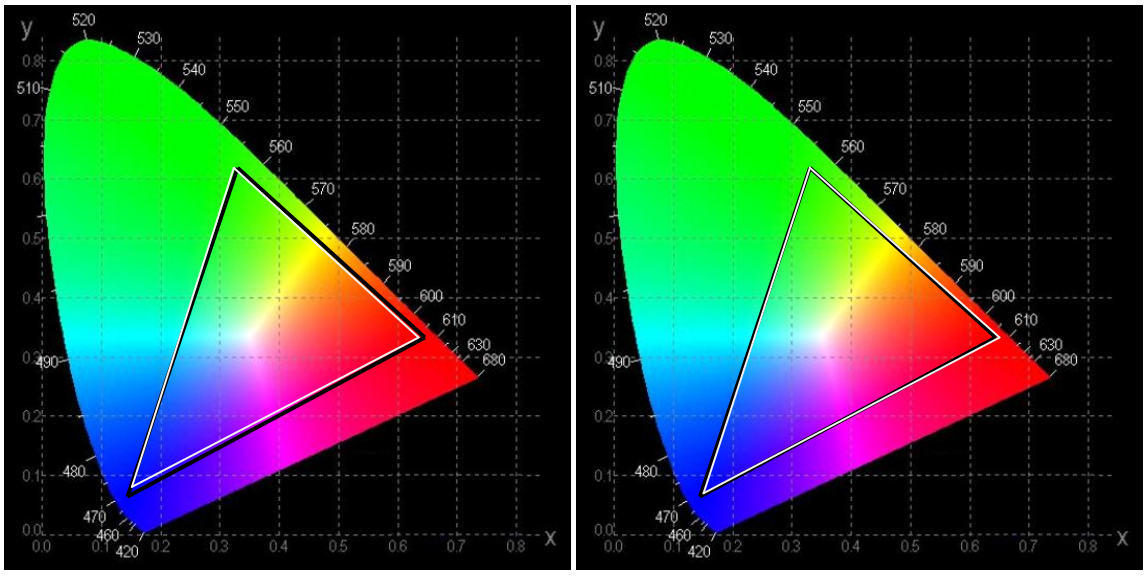
Neben dem Leuchtkraftverlust kann es durch Vorrichtungen, die zwischen Augen und Projektionsfläche geschaltet werden, auch zu Veränderungen der dargestellten Farben kommen (Veränderungen des Gamut). Wie stark diese Änderungen ausfallen lässt sich messen, indem in einem ersten Schritt der Gamut der Projektionsfläche ohne Vorrichtung bestimmt wird. Dazu werden die Farben Rot, Grün und Blau auf der Projektionsfläche dargestellt und mit einem Chromameter gemessen. In einem zweiten Schritt wird die Vorrichtung zum Trennen von linkem und rechtem Bild, vor dem Chromameter platziert und die Messungen erneut durchgeführt. Trägt man diese Werte in den CIE-XYZ-Farbraum ein und verbindet die zusammengehörenden Rot-, Grün- und Blauwerte zu Dreiecken, kann man die Farbveränderungen des Gamut ablesen. Je geringer der Unterschied zwischen den Dreiecken ausfällt desto besser die Vorrichtung.

In Abbildung 6 wird die Veränderung des Gamut durch zwei unterschiedliche Polarisationsfilter sowie durch NuVision 60GX Shutterglasses dargestellt.

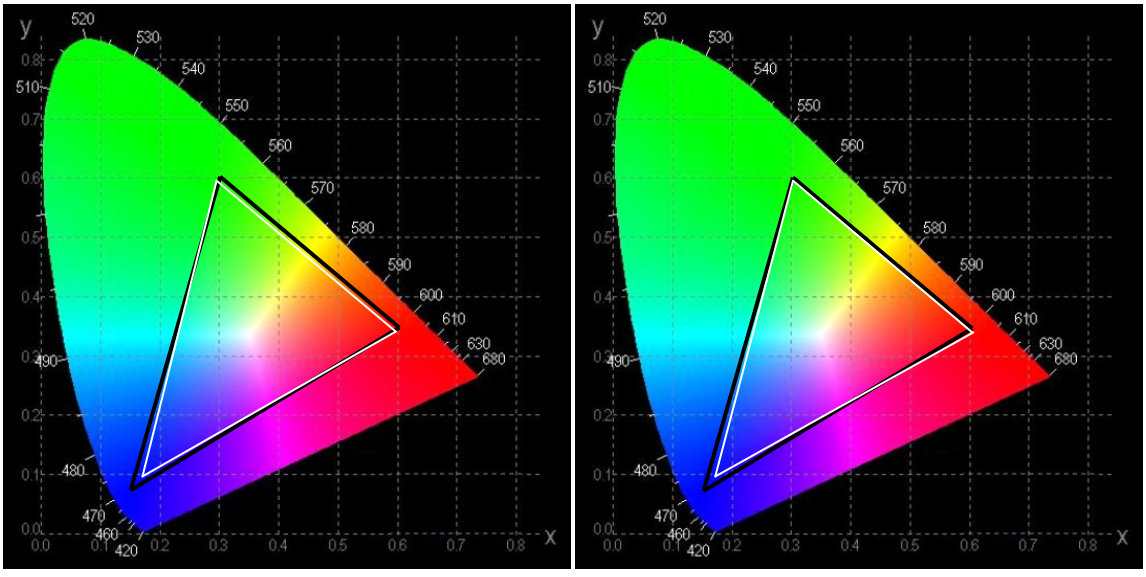
a)


y
x
520
530
540
550
560
570
580
590
600
610
630
680
490
510
480
470
460
420

b)


y
x
520
530
540
550
560
570
580
590
600
610
630
680
490
510
480
470
460
420

c)


y
x
520
530
540
550
560
570
580
590
600
610
630
680
490
510
480
470
460
420

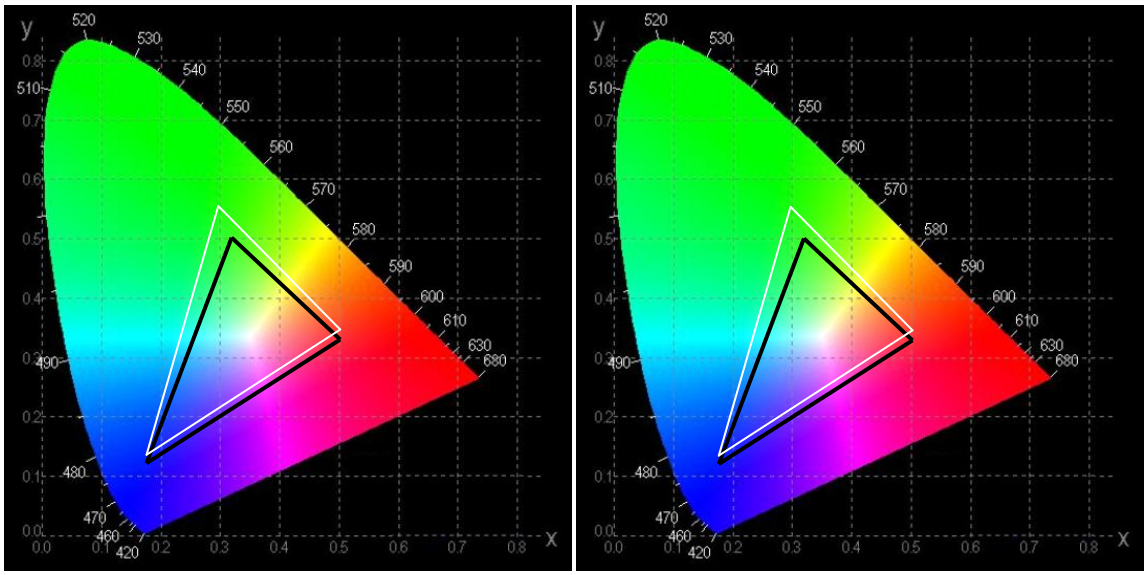
d) 

Abbildung 6: Die Veränderung des Gamut einer Projektionsfläche durch eine Vorrichtung zum Trennen von linken und rechten Bild (Polarisationsfilter, Shutterglasses). Die schwarzen Dreiecke beschreiben den Gamut der Projektionsfläche und die weißen Dreiecke den durch die Polarisationsfilter veränderten Gamut. a) Polarisationsfilter Captain. b) Polarisationsfilter Captain Spezialanfertigung. c) Polarisationsfilter Real3D. d) Shutterglasses NuVision G0GX.

Referenzen

[1] Herbert Grasberger, „Introduction to Stereo Rendering“, TU Wien, 2008

[2] http://stereo.jpn.org/eng/stvply/index.html

[3] http://www.nvidia.com/object/GeForce_3D_Drivers_Downloads.html

[4] http://www.deskshare.com/msr.aspx

[5] http://camstudio.org/

[6] http://www.fraps.com/

[7] „3D Stereoscopic Game Development“, Presentation at GDC, nVidia Corporation, Santa Clara, 2008

[8] eMagin Corporation, „eMagin SDK Version 2.2“, 2006

[9] Dave Shreiner, Mason Woo, Jackie Neider , Tom Davis, „*OpenGL Programming Guide Fifth Edition*“, p. 464, 2006

[10] http://www.nvidia.com/object/3D_Vision_Discover_requirements.html

[11] Kuo-Chung HuangI, Chao-Hsu Tsai, Kuen Lee, Wen-Jean Hsueh „Measurement of Contrast Ratios for 3D Display“, *Input/Output and Imaging Technologies II.*, Taipei, Taiwan, 26-27, 2000