Computergrafik Praktikum 1

Allgemeines zum Praktikum

Im Rahmen des Computergrafik-Praktikums lernen Sie, die aus der Vorlesung gewonnenen Erkenntnisse praktisch mit der Programmiersprache C/C++ umzusetzen.

Bei der Konzipierung der Praktika wurde darauf geachtet, dass die verwendeten Bibliotheken sowohl für Windows als auch für OS X erhältlich sind. Sofern Sie die Aufgaben im Medienlabor bearbeiten, steht Ihnen die Entwicklungsumgebung XCode zur Verfügung, mit der Sie Ihre C++-Programme entwickeln können. Wenn Sie die Aufgaben an ihrem privaten Rechner unter Windows bearbeiten möchten, empfehle ich Ihnen die Entwicklungsumgebung Visual Studio 2013, die Sie als Student/-in über das Microsoft ELMS-Programm kostenlos herunterladen können.

Bitte beachten Sie bei der Bearbeitung der Praktikumsaufgaben, dass Ihre Lösungen für folgende Praktikumsaufgaben weiter genutzt werden müssen. Die Praktikumsarbeiten bauen also ineinander auf. Bearbeiten Sie deshalb die Lösungen sorgfältig und löschen Sie diese nicht, wenn die Aufgabe abgenommen wurde. Achten Sie bei der Entwicklung darauf, dass nur die Schnittstellen in den Header-Dateien stehen, während die Implementierung ausschließlich in den cpp-Dateien steht.

Zum Bestehen des kompletten Praktikums müssen Sie jede Praktikumsaufgabe erfolgreich abschließen und mindestens 80% der Gesamtpunkte für das komplette Praktikum erhalten. Wie viele Punkte Sie für eine einzelne Aufgabe erhalten, hängt von Ihrer Implementierung und Ihren Erläuterungen ab.

Um die Aufgaben dieses Aufgabenzettels zu bestehen, müssen alle mitgelieferten Test-Klassen erfolgreich durchlaufen (5 Punkte) und Sie müssen die einzelnen entwickelten Methoden erläutern können (weitere 5 Punkte). Insgesamt können Sie bei diesem Aufgabenblatt 10 Punkte erhalten.

Abnahmetermin: 29.03. (Dienstagsgruppe) & 01.04. (Freitagsgruppen)

Thema Praktikum 1

Ziel der ersten beiden Praktikumsaufgaben (Praktikum 1 & 2) ist es, einen einfachen Raytracer zur Offline-Bildsynthese zu programmieren. Hierfür entwickeln wir im ersten Schritt (Praktikum 1) drei Hilfsklassen (Vector, Color, RGBImage). Die Klassen sollen uns die Arbeit mit Vektoren und Farben erleichtern, sowie eine Möglichkeit zur Speicherung eines Bildes bereitstellen.

Sie finden die Dateien mit den Schnittstellen für die Vector-, Color- & RGBImage-Klasse und die Testklassen im OSCA-Lernraum im Archiv CGPrakt1.zip. In dem Archiv befinden sich ebenfalls eine Xcode- und Visual-Studio-Projektdatei.

Aufgabe 1 (5 Punkte)

Implementieren Sie die folgende Vektor-Klasse (die Dateien Vector.h und Vector.cpp finden Sie im Archiv CGPrakt1.zip):

```
class Vector
{
public:
    float X;
    float Y;
    float Z;

    Vector( float x, float y, float z);
    Vector();

    float dot(const Vector& v) const;
    Vector cross(const Vector& v) const;
    Vector operator+(const Vector& v) const;
    Vector operator-(const Vector& v) const;
    Vector& operator+=(const Vector& v);
    Vector operator*(float c) const;
    Vector operator-() const;
    Vector& normalize();
    float length() const;
    float lengthSquared() const;
    Vector reflection( const Vector& normal) const;
    bool triangleIntersection( const Vector& d, const Vector& a, const Vector& b,
                               const Vector& c, float& s) const;
};
```

Erläuterungen:

- Der Standardkonstruktor lässt die Member-Variablen X,Y,Z uninitialisiert, während der Konstruktor mit Parametern X=x, Y=y & Z=z setzt.
- Die Methode *dot(..)* berechnet das Skalarprodukt.
- Die Methode *cross(..)* berechnet das Kreuzprodukt.
- Die Methode *normalize()* normalisiert den Vektor.
- Die Methode *length()* gibt die Länge des Vektors zurück (die Methode sollte auf *lengthSquared* zurückgreifen, nicht anders herum, warum?).
- Die Methode *lengthSquared()* gibt die quadratische Länge des Vektors zurück.
- Die Methode *reflection(...)* liefert den Reflexionsvektor zurück. *normal* ist die Normale der Oberfläche, die reflektiert.
- *triangleIntersection* führt einen Kollisionstest zwischen einem Strahl und einem Dreieck durch. D bezeichnet die Richtung des Strahls und der Ursprung des Strahls ist (*this). Das Dreieck ist durch dessen drei Eckpunkte (a,b,c) definiert. Der Parameter s wird von der Methode beschrieben und erlaubt die Bestimmung des Auftreffpunkts, indem folgende Berechnung durchgeführt wird: Auftreffpunkt = (*this) + d*s.

Testen Sie Ihre Implementierung mit dem Aufruf der Methode *Test1::vector()*. Die Testklasse können Sie im Lernraum herunterladen (CGPrakt1.zip).

Tipps:

- Denken Sie daran, dass Sie in C++ nicht alle Objekte mit *new* erzeugen müssen. Objekte, die mit *new* erzeugt wurden, werden dauerhaft auf dem Heap abgelegt und müssen mit *delete* wieder gelöscht werden. Die Vektorklasse benötigt keine Elemente auf dem Heap, von daher wird auch kein *new* benötigt.

- Bedenken Sie bei der Implementierung der *triangleIntersection()*-Methode, dass der Strahl nur in eine Richtung verläuft. Die Testklasse prüft, ob die Methode auch eine Kollision erkennt, wenn der Strahl vom Dreieck weg zeigt. In diesem Fall sollte die Methode keine Kollision melden.
- Die in der Vorlesung besprochene Methode zur Kollisionserkennung zwischen einem Strahl und einem Dreieck prüft, ob die Summe der Teildreiecke kleiner gleich der Dreiecksfläche ist (abc>=abp+acp+bcp). Bedenken Sie, dass Ungenauigkeiten beim Floating-Point-Typ dazu führen können, dass die Flächen nicht identisch sind. Deshalb sollten Sie ein kleines Epsilon definieren, so dass abc+Epsilon >= abp+acp+bcp. Die Testklassen verwenden Epsilon=1e-6.

Aufgabe 2 (2 Punkte)

Implementieren Sie die folgende Color-Klasse (die Dateien Color.h und Color.cpp finden Sie im Archiv CGPrakt1.zip):

```
class Color
{
public:
    float R;
    float G;
    float B;

    Color();
    Color( float r, float g, float b);
    Color operator*(const Color& c) const;
    Color operator*(const float Factor) const;
    Color operator+(const Color& c) const;
    Color& operator+=(const Color& c);

};
```

Erläuterungen:

- Die Klasse soll einen Farbwert durch drei Float-Werte für Rot (R), Grün (G) und Blau (B) repräsentieren.
- Beim Aufruf des Standardkonstruktors soll die Farbe schwarz gesetzt werden, beim Aufruf des Konstruktors mit Parametern sollen die entsprechenden übergebenen Farben gesetzt werden.

Testen Sie Ihre Implementierung mit dem Aufruf der Methode *Test2::color()*. Die Testklasse können Sie im Lernraum herunterladen (CGPrakt1.zip).

Aufgabe 3 (3 Punkte)

Implementieren Sie die folgende RGBImage-Klasse (die Dateien rgbimage.h und rgbimage.cpp finden Sie im Archiv CGPrakt1.zip)):

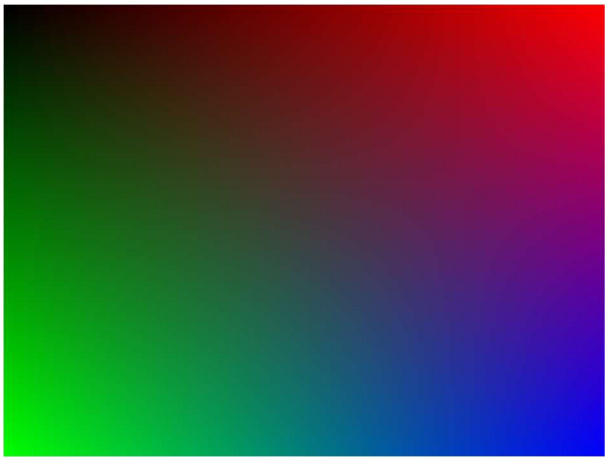
```
class RGBImage
{
public:
    RGBImage( unsigned int Width, unsigned Height);
    ~RGBImage();
    void setPixelColor( unsigned int x, unsigned y, const Color& c);
    const Color& getPixelColor( unsigned int x, unsigned y) const;
    bool saveToDisk( const char* Filename);
    unsigned int width() const;
    unsigned int height() const;

    static unsigned char convertColorChannel( float f);
protected:
    Color* m_Image;
    unsigned int m_Height;
    unsigned int m_Width;


};
```

Erläuterungen:

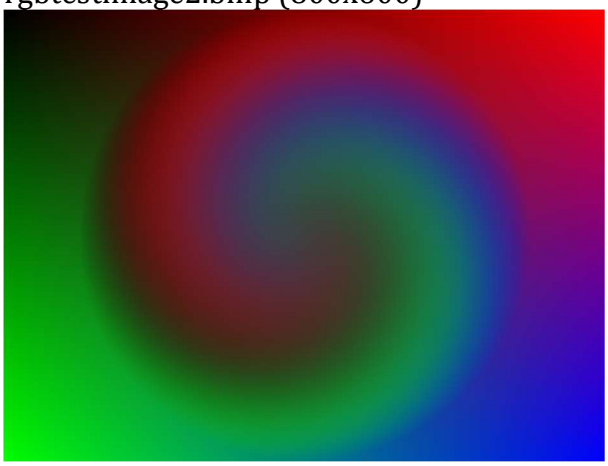
- Die Klasse erzeugt bei der Konstruktion einen 2D-Bildspeicher mit der Auflösung Width*Height Pixeln. Im Destruktor wird der Bildspeicher wieder freigegeben.
- Über *setPixelColor* & *getPixelColor* kann eine Pixelfarbe an der Stelle x,y (Pixelkoordinaten) abgefragt werden. Stellen Sie für das Setzen und Lesen der Pixelfarben sicher, dass keine ungültigen Bereiche des Bildes (Bereiche außerhalb von *width()* bzw. *height()*) beschrieben bzw. abgefragt werden.
- Das Bild soll zeilenweise organisiert sein, das bedeutet, die Koordinate x=0, y=0 liefert den oberen, linken Pixel zurück, x=*width()*-1 & y=0 liefert den oberen, rechten Pixel zurück, x=0 & y=*height()*-1 liefert den unteren, linken Pixel zurück, x=*width()*-1 & y=*height()*-1 liefert den unteren rechten Pixel zurück.
- Die statische Methode *convertColorChannel(..)* konvertiert einen Float-Farbkanal (R, G oder B) in einen Byte-Wert. Für die Konvertierung gilt: 0.0f ➔0, 1.0f➔255. Ist f < 0.0f, dann wird f=0.0f angenommen und bei f>1.0f wird f=1.0f angenommen.
- Die Methode *saveToDisk(...)* soll das Bild als 24Bit-RGB-Bitmap-Datei (BMP) speichern. Nutzen Sie die *convertColorChannel()*-Methode, um die Farben zu konvertieren. Zum Öffnen und Beschreiben einer Datei können die C-Funktionen *fopen(..)*, *fwrite(..)* und *fclose(..)* genutzt werden. Die Funktionen sind in der *stdio.h* definiert. Eine Beschreibung des BMP-Dateiformats finden Sie unter http://de.wikipedia.org/wiki/Windows_Bitmap .

Testen Sie Ihre Implementierung mit dem Aufruf der Methode Test3::rgbimage(..). Die Testklasse können Sie im Lernraum herunterladen (CGPrakt1.zip). Die Methode erwartet beim Aufruf die Angabe eines existierenden Verzeichnisses, in dem die Testbilder gespeichert werden. Nach dem Durchlauf der Test-Routine sollten in dem Verzeichnis drei BMP-Dateien liegen („rgbtestimage1.bmp“, ..., „rgbtestimage3.bmp“), die wie folgt aussehen sollten:

rgbtestimage1.bmp (800x600)



rgbtestimage2.bmp (800x600)



rgbtestimage3.bmp (800x600)