Programmierübung 3: Rasterung
Computergrafik, Herbst 2012

Abgabedatum: Donnerstag 1. November, 11:59

Diese Übung muss bis Donnerstag 1. November um 12:00 Uhr via Ilias abgegeben werden. Zusätzlich muss die Übung wie üblich einem Assistenten am Computer gezeigt werden. Schreiben Sie sich dazu wieder auf der Online Liste ein.

Erstellen Sie für die einzelnen Teilaufgaben separate Eclipse Projekte, damit Sie diese getrennt demonstrieren können. Sie können jeweils das Projekt "simple" kopieren und dieses dann erweitern.

Bei dieser Übung geht es darum, einen eigenen Software Rasterer zu implementieren. Der zur Verfügung gestellte Code ist bereits darauf ausgelegt, dass der bisherige OpenGL Renderer durch einen Software Renderer, der vollständig in Java implementiert ist, ersetzt werden kann. Leiten Sie dazu Ihre `RenderPanel` Implementation (Bspw. `SimpleRenderPanel`) von `jrtr.SWRenderPanel` statt von `jrtr.GLRenderPanel` ab. Wenn Sie die Applikation nun starten, sollte ein schwarzes Fenster dargestellt werden. Damit sind Sie bereit, mit Ihrer Implementation zu beginnen.

1 Rendern von Dreieckspunkten (2 Punkte)

In einem ersten Schritt sollen die Eckpunkte der Dreiecke korrekt projeziert und an der richtigen Stelle im 2D Bild ausgegeben werden. Implementieren Sie dazu die Methode `SWRenderContext.draw`, so dass jeder Eckpunkt in Bildkoordinaten umgerechnet wird und setzen Sie die Farbe des entsprechenden Pixels auf weiss.

Das Ausgabebild wird in einem Objekt des Typs `java.awt.image.BufferedImage` gespeichert und kann über die Klassenvariable `colorBuffer` angesprochen werden. Mit der Methode `BufferedImage.setRGB` können Sie den Farbwert eines einzelnen Pixels setzen. Bei der Viewport Transformation ist zu beachten, dass Pixel 0,0 im `BufferedImage` in der linken oberen statt in der linken unteren Ecke liegt.

Rendern Sie einige Testbilder und vergleichen Sie diese mit der OpenGL Version in dem Sie das `SWRenderPanel` durch das `GLRenderPanel` ersetzen.

2 Rasterung und Z-Buffering (4 Punkte)

Implementieren Sie das in der Vorlesung vorgestellte Verfahren zur Rasterung von Dreicken mit homogenen 2D-Koordinaten. Verwenden Sie einen Z-Buffer um das Sichtbarkeitsproblem zu lösen. Um einzelne Dreiecke zu zeichnen, müssen zuerst die Daten von jeweils drei Eckpunkten aus der `VertexData` Struktur gesammelt werden. Dies kann wie in diesem Codefragment angedeutet implementiert werden:

```
// Variable declarations
VertexData vertexData;
LinkedList<VertexData.VertexElement>;
int indices[];
float[][] colors;
Matrix4f t;

// Skeleton code to assemble triangle data
int k = 0; // index of triangle vertex, k is 0,1, or 2

// Loop over all vertex indices
for(int j=0; j<indices.length; j++)
{
  int i = indices[j];

  // Loop over all attributes of current vertex
  ListIterator<VertexData.VertexElement> itr =
    vertexElements.listIterator(0);
  while(itr.hasNext())
  {
    VertexData.VertexElement e = itr.next();
    if(e.getSemantic() == VertexData.Semantic.POSITION)
    {
      Vector4f p = new Vector4f
        (e.getData()[i*3],e.getData()[i*3+1],e.getData()[i*3+2],1);
      t.transform(p);
      positions[k][0] = p.x;
      positions[k][1] = p.y;
      positions[k][2] = p.z;
      positions[k][3] = p.w;
      k++;
    }
    else if(e.getSemantic() == ...
    // you need to collect other vertex attributes (colors, normals) too

    // Draw triangle as soon as we collected the data for 3 vertices
```

```
    if(k == 3)
    {
      // Draw the triangle with the collected three vertex positions, etc.
      rasterizeTriangle(positions, colors, normals, ...);
      k = 0;
    }
  }
}
```

Bestimmen Sie beim Rastern eines Dreiecks zuerst dessen Bounding Box beschränkt auf das 2D Bild wie in der Vorlesung besprochen. Innerhalb der Bounding Box berechnen Sie für jedes Pixel die Kantenfunktionen und ermitteln so ob sich das Pixel innerhalb des Dreiecks befindet. Mittels Z-Buffer stellen Sie nun fest ob das Pixel gezeichnet werden soll. Verwenden Sie für den Z-Buffer den Wert von $1/w$. Ermitteln Sie zu jedem Pixel, das gezeichnet werden soll, die perspektivisch korrekt interpolierte Farbe aus den Farben der 3 Eckpunkte des Dreiecks.

3 Texture Mapping (4 Punkte)

Das Ziel dieser Aufgabe ist es, Ihre Software Rasterung um Texture Mapping zu erweitern. Dazu müssen Sie zuerst einige Vorbereitungen treffen. Implementieren Sie in der Klasse `jrtr.SWTexture` die Methode `load`, welche ein Bild als Textur lädt. Das kann mit folgendem Code erreicht werden:

```
BufferedImage texture;
File f = new File(fileName);
texture = ImageIO.read(f);
```

Auf die Pixel der Textur können Sie später über die Methoden von `BufferedImage` zugreifen, ähnlich wie Sie das im Software Rasterer tun.

Es ist nützlich, Texturen einzelnen Objekten in der Szene zuweisen zu können. Zu diesem Zweck statten Sie am besten die Klasse `Material` mit einem Zeiger auf eine `Texture`, und die Klasse `Shape` mit einem Zeiger auf ein `Material` aus. Später werden wir dann die `Material` Klasse um weitere Attribute erweitern. Wenn ein `Shape` gerastert wird, kann nun also via das `Material` auf die `Texture` zugegriffen werden.

Erweitern Sie Ihren Rasterer so, dass an jedem Pixel die Texturkoordinaten perspektivisch korrekt interpoliert werden. Die Texturkoordinaten werden dann verwendet, um den entsprechenden Farbwert aus der Textur zu lesen und dem Pixel zuzuweisen. Implementieren Sie zuerst Nearest-Neighbor Interpolation, und nachher auch bilineare Interpolation.

Um mit Texturen zu rendern, müssen natürlich Ihre Objekte mit passenden Texturkoordinaten versehen sein. Ein Quadrat (bestehend aus 2 Dreicken) mit Texturkoordinaten kann z.B. folgendermassen erstellt werden:

```
float v[] = {-1,-1,1, 1,-1,1, 1,1,1, -1,1,1};
```

```
float t[] = {0,0, 1,0, 1,1, 0,1};
VertexData vertexData = new VertexData(4);
vertexData.addElement(v, VertexData.Semantic.POSITION, 3);
vertexData.addElement(t, VertexData.Semantic.TEXCOORD, 2);
int indices[] = {0,2,3, 0,1,2};
vertexData.addIndices(indices);
```

Beachten Sie die Konvention, dass Texturkoordinaten (0,0) der linken untere Ecke, und (1,1) der rechten oberen Ecke der Textur entsprechen sollen. Das heisst, Sie müssen die interpolierten Texturkoordinaten beim Zugriff auf die Textur entsprechend umrechnen.

Demonstrieren Sie Ihre Texture Mapping Funktionalität mit einer passenden Szene. Versehen Sie dazu Ihr Torusgitter, Ihr Zylindergitter, die vorgegebene Szene mit dem Haus aus der letzten Übung, oder die fraktale Landschaft mit Texturkoordinaten.