Kurzeinführung in OpenGL(ES)

Matthias Braun
matthias.braun@kit.edu

Institut für Programmstrukturen und Datenorganisation
Lehrstuhl für Programmierparadigmen

Wintersemester 2009/10

Was ist OpenGL?

Open Graphics Library (OpenGL)

- Ist eine Bibliothek zur Ansteuerung von Grafikhardware
- Platform und Programmiersprachenunabhängig
- Darstellung komplexer 3D-Szenen in Echtzeit
- Heute Standardschnittstelle auf praktisch allen Platformen mit 3D-Unterstützung. (Auf Microsoft-Systemen allerdings Dominanz von Direct-3D)

Was ist OpenGL ES?

Open Graphics Library for Embedded Systems (OpenGL ES)

- Abgespeckte Version von OpenGL für eingebettete Systeme
- Entfernung von Altlasten (`glBegin`, `glEnd`)
- Beschränkung der Datentypen (kein `double`)
- Einsatz zum Beispiel auf iPhone, Android, Playstation 3.

Zeichenbereich — der OpenGL-Kontext

- Ein OpenGL-Kontext ist ein 2-dimensionaler rechteckiger Bereich. Typischerweise ein Programmfenster oder der ganze Bildschirm.
- Die Bereitstellung eines Kontextes ist Architekturabhängig.
- OpenGL ist nur zum Zeichnen auf den Kontext verantwortlich. Dinge wie Tastatur, Touchpad oder Zeitmessung werden anderen Bibliotheken überlassen.
- Typischerweise werden Double-Buffering Techniken verwendet:
  - Die Szene wird in einen nicht-sichtbaren Puffer gezeichnet.
  - Dieser Puffer wird danach auf dem Bildschirm kopiert.
    $\Rightarrow$ Dies vermeidet Flackern da Zwischenschritte beim Bildaufbau nicht gezeigt werden.
- Innerhalb des Kontexts können rechteckige Bereiche (*Viewports*) zum Zeichnen ausgewählt werden.

Android OpenGL-Kontext

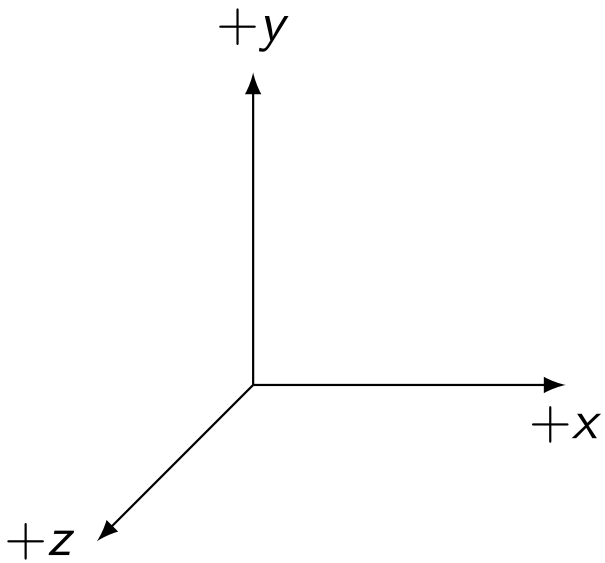
Klasse `GLSurfaceView.Renderer` implementieren!

```
public abstract interface Renderer {
    /** Wird aufgerufen wenn ein neuer Zeichenbereich
     * erzeugt wurde */
    void onSurfaceCreated(GL10 gl, EGLConfig config);
    /** Wird aufgerufen wenn sich der Zeichenbereich
     * geaendert hat */
    void onSurfaceChanged(GL10 gl, int width, int height);
    /** Wird aufgerufen wenn ein neues Bild gezeichnet
     * werden soll */
    void onDrawFrame(GL10 gl);
}
```

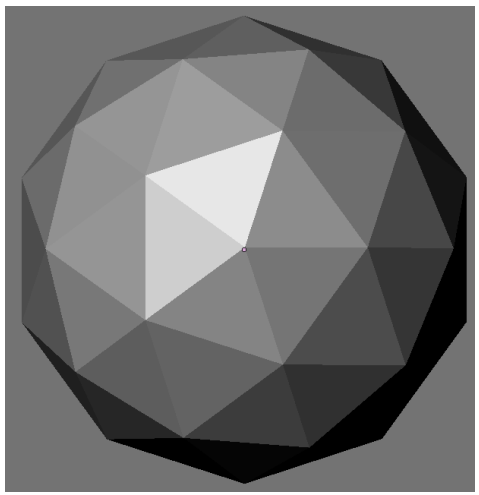
J2ME OpenGL-Kontext

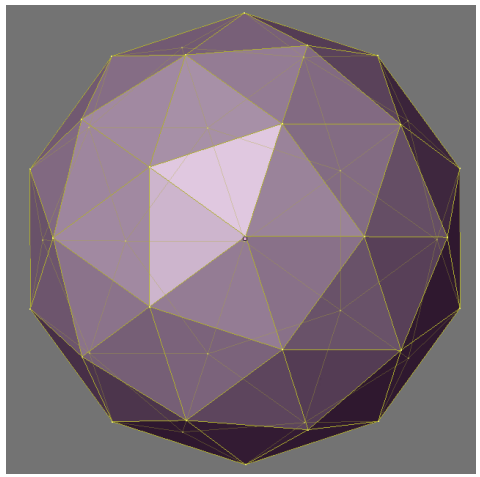
siehe `apps/OpenGLESDemo` im Java Wireless Toolkit.

3D-Szenen und Geometrie



- 3D-Geometrie wird in einem kartesischen Koordinatensystem angegeben.
- Das einzige Primitiv das gezeichnet werden kann ist das Dreieck. Würfel, Kugeln, etc. werden stets aus Dreiecken zusammengesetzt.

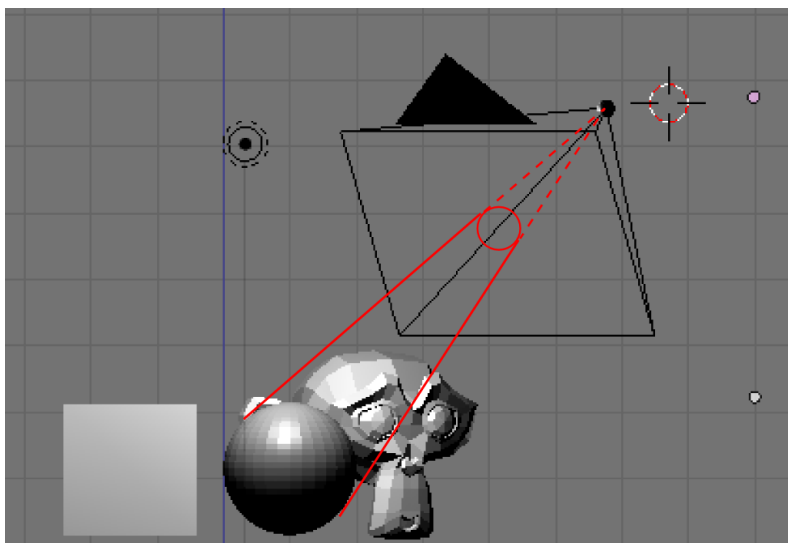


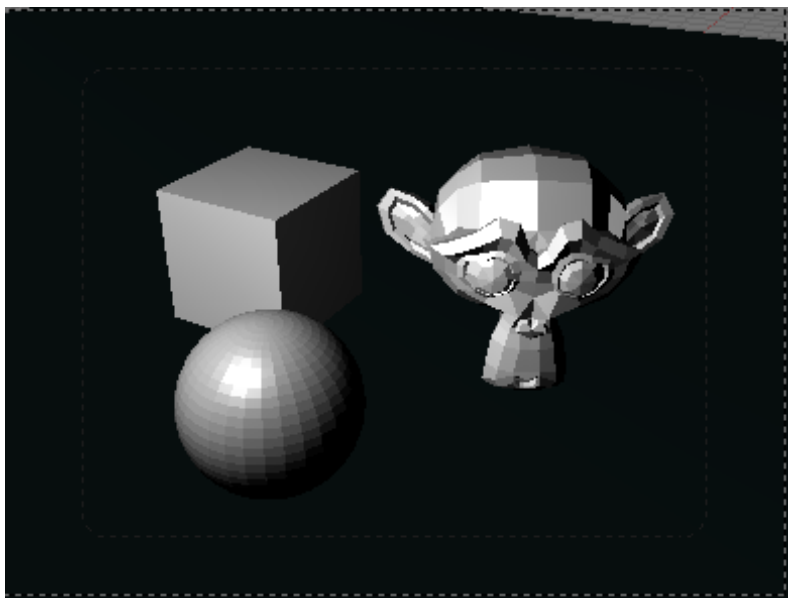


3D-Projektion

Wie bringt man 3-dimensionale Geometrie auf einen 2-dimensionalen Bildschirm?

Perspektivische Projektion: Sehstrahlen gehen von einem Punkt (Kamera/Auge) aus. Kurz vor diesem befindet sich die 2-dimensionale Bildfläche.





Projektion in OpenGL

Die Projektion in OpenGL wird durch die `PROJECTION`-Matrix bestimmt. Diese setzt man am einfachsten mit `glFrustum`:

```
glFrustumf(left, right, bottom, top, zNear, zFar)
```

Beispiel:

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
float ratio = screen_width / screen_height;
glFrustumf(-ratio, ratio, -1, 1, 1, 10);
```

OpenGL Zustand

OpenGL ist eine Zustandsmaschine. Tyische Modi die man zu Begin des Programs setzt:

```
glEnable(GL_DEPTH_TEST);
// glEnable(GL_CULL_FACE);
// glEnable(GL_BLEND);
// glShadeModel(GL_SMOOTH);
// glEnable(GL_TEXTURE_2D);
glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_FASTEST);
```

OpenGL Viewport Initialisieren

Der Viewport muss am Anfang gesetzt werden und jedes Mal wenn sich die Größe des Fensters/Bildschirms ändert.

```
/* viewport consists of the whole screen */
glViewPort(0, 0, width, height);

/* setup projection matrix */
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
float ratio = screen_width / screen_height;
glFrustumf(-ratio, ratio, -1, 1, 1, 10);
```

Typischer Ablauf

- Kontext und Viewport initialisieren
- Projektion einstellen
- Wiederhole
  - `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)`
  - Kameratransformation setzen
  - Objekte zeichnen
  - (`glFlush()`)

Transformationen sind Matrizen

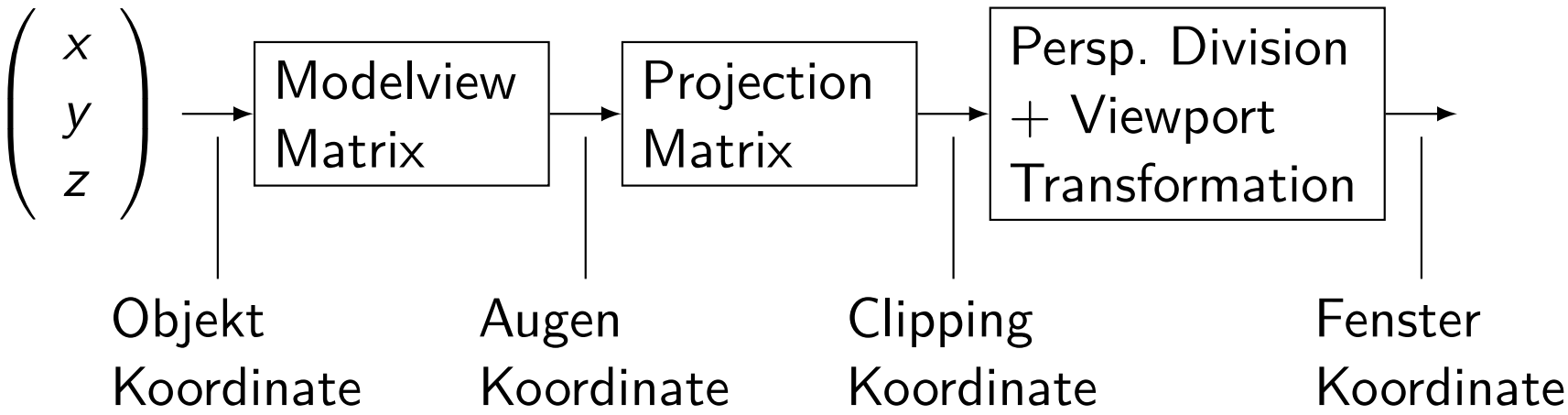
Typische Transformationen für 3D-Geometrie sind:

- *Verschieben* (`glTranslatef(x, y, z)`)
- *Skalieren* (`glScalef(x, y, z)`)
- *Rotieren* (`glRotatef(angle_degree, x, y, z)`)

Dies sind alles lineare Abbildungen und lassen sich deshalb einheitlich in einer *Matrix* repräsentieren.
Mehrere Transformationen lassen sich kombinieren indem man ihre entsprechenden Matrizen multipliziert!

$$\text{translate}(\text{scale}(v)) = M_{\text{translate}} \cdot M_{\text{scale}} \cdot v$$

Grafik Pipeline



Matrixmanipulation

- Matrix wählen: `glMatrixMode(matrix)`
  matrix ist `GL_MODELVIEW`, `GL_PROJECTION` oder `GL_TEXTURE`
- Einheitsmatrix setzen: `glLoadIdentity()`
- Mit Rotationsmatrix multiplizieren:
  `glRotatef(angle, x, y, z)`
- Mit Skalierungsmatrix multiplizieren: `glScalef(x, y, z)`
- Mit Verschiebematrix multiplizieren:
  `glTransformf(x, y, z)`
- Matrixstack (siehe nächste Folien): `glPushMatrix()`, `glPopMatrix()`
- Weitere Funktionen: `glLoadMatrix`, `glMultMatrix`, `glGet`

Beispiel 1: „Kamera bewegen und um Ursprung rotieren”

```
/* ModelviewMatrix neu setzen */
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
/* Kamera 5 Einheiten vom Ursprung wegbewegen */
glTranslatef(0, 0, -5);
/* Um "angle" grad nach rechts um den Ursprung rotieren */
glRotatef(angle, 0, 1, 0);

/* Szene Zeichnen */
drawEarth();
```

Live Demo

Beispiel 2: „Erde umkreist Sonne”

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0, 0, -5);

drawSun();

glRotatef(angle, 0, 1, 0);
glTranslatef(2f, 0, 0);
glScalef(0.6f, 0.6f, 0.6f);
drawEarth();
```

Live Demo

Der Matrixstack

Häufig will man Transformationen nur auf einen Teil der Szene anwenden. Beispielsweise will man die letzten glRotatef, glTranslatef, glScalef nur auf die „Erde" Anwenden, nicht auf danach gezeichnete weitere Planeten. Dafür kann man Matrizen auf den OpenGL Matrixstack legen:

- `glPushMatrix()` legt aktuelle Matrix auf Matrixstack
- `glPopMatrix()` läd oberste Matrix vom Matrixstack

Beispiel3: „Sonne, Erde und Mars”

```
/* ... */
drawSun();

glPushMatrix();
glRotatef(angle, 0, 1, 0);
glTranslatef(2f, 0, 0);
glScalef(0.6f, 0.6f, 0.6f);
drawEarth();
glPopMatrix();

glPushMatrix();
glRotatef(angle*1.6f, 0, 1, 0);
glTranslatef(0, 0, 3f);
glScalef(0.3f, 0.3f, 0.3f);
drawMars();
glPopMatrix();
```

Live Demo

Typischer Ablauf 2

- Kontext und Viewport initialisieren
- Projektion einstellen
- Wiederhole
  - `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)`
  - Kameratransformation setzen
  - Für alle Objekte (auch rekursiv)
    - `glPushMatrix()`
    - Objekttransformation setzen
    - Objekt zeichnen
    - `glPopMatrix()`
  - (`glFlush()`)

Geometrie erzeugen

Um Dreiecke zu zeichnen muss man 2 Dinge angeben:

- Eine Liste mit Punkten (Vertices)
- Eine Liste die angibt welche Punkte zu dreiecken verbunden werden

Implementierung:

- Erzeuge float[] Reihung. Jeweils 3 aufeinanderfolgende Elemente ergeben einen (x, y, z) Vektor (genannt VertexArray).
- Erzeuge short[] Reihung. Elemente sind Indizes ins VertexArray (genannt IndexBuffer). [1]
- Zeichnen:

```
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3, GL_FLOAT, 0, vertexArray);
glDrawElements(GL_TRIANGLES, n_triangles,
                    GL_UNSIGNED_SHORT, indexBuffer);
```

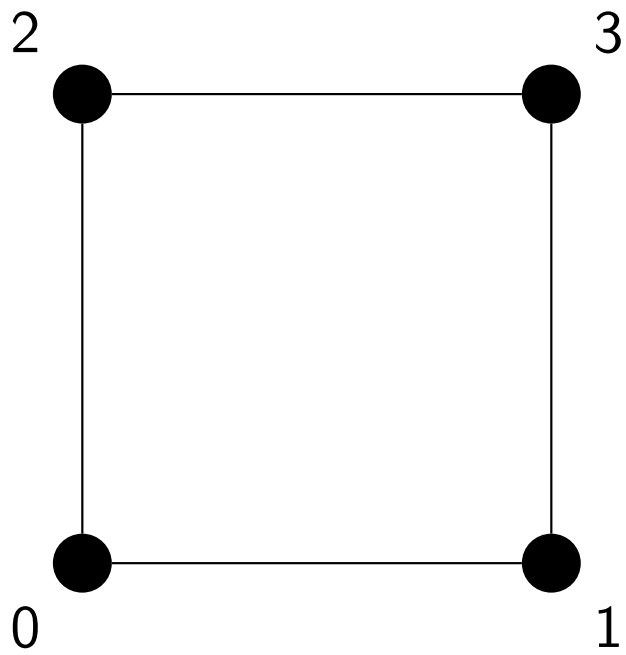
[1] Die Indizes beziehen sich auf Punkte nicht auf Einträge im VertexArray
$\Rightarrow i_{\text{VertexArray}} = i * 3$

Puffer in Android

OpenGL ist eine Hardwarenahe Bibliothek und benötigt Puffer die unabhängig von der Java Garbage-Collection arbeiten. Abhilfe:

```
private static FloatBuffer makeDirectFloatBuffer(float[] array) {
    int len = array.length * (Float.SIZE/8);
    ByteBuffer storage = ByteBuffer.allocateDirect(len);
    storage.order(ByteOrder.nativeOrder());
    FloatBuffer buffer = storage.asFloatBuffer();
    buffer.put(array);
    buffer.position(0);
    return buffer;
}
private static ShortBuffer makeDirectShortBuffer(short[] array) {
    int len = array.length * (Short.SIZE/8);
    ByteBuffer storage = ByteBuffer.allocateDirect(len);
    storage.order(ByteOrder.nativeOrder());
    ShortBuffer buffer = storage.asShortBuffer();
    buffer.put(array);
    buffer.position(0);
    return buffer;
}
```

Beispiel: Rechteck




```
float[] rectVertices = new float[] {
        −1, −1, 0,
         1, −1, 0,
         1, 1, 0,
        −1, 1, 0
};
rectVerticesBuffer
        = makeDirectFloatBuffer(rectVertices);

short[] rectTriangles = new short[] {
        0, 1, 2,
        2, 3, 0
};
rectTrianglesBuffer
        = makeDirectShortBuffer(rectTriangles);
```

Beispiel: Würfel

```
float[] vertices = new float[] {
    −1, −1, −1,
    −1, 1, −1,
     1, 1, −1,
     1, −1, −1,
    −1, −1, 1,
    −1, 1, 1,
     1, 1, 1,
     1, −1, 1
};
verticesBuffer
    = makeDirectFloatBuffer(vertices);
```

```
short[] triangles = new short[] {
    /* front */
    0, 1, 2,
    2, 3, 0,
    /* back */
    6, 5, 4,
    4, 7, 6,
    /* top */
    4, 5, 1,
    1, 0, 4,
    /* bottom */
    2, 6, 7,
    7, 3, 2,
    /* left */
    7, 4, 0,
    0, 3, 7,
    /* right */
    1, 5, 6,
    6, 2, 1
};
```

Farben

- Farben werden als 4-Tupel angegeben:

$$(\text{rot}, \text{grün}, \text{blau}, \text{alpha})$$

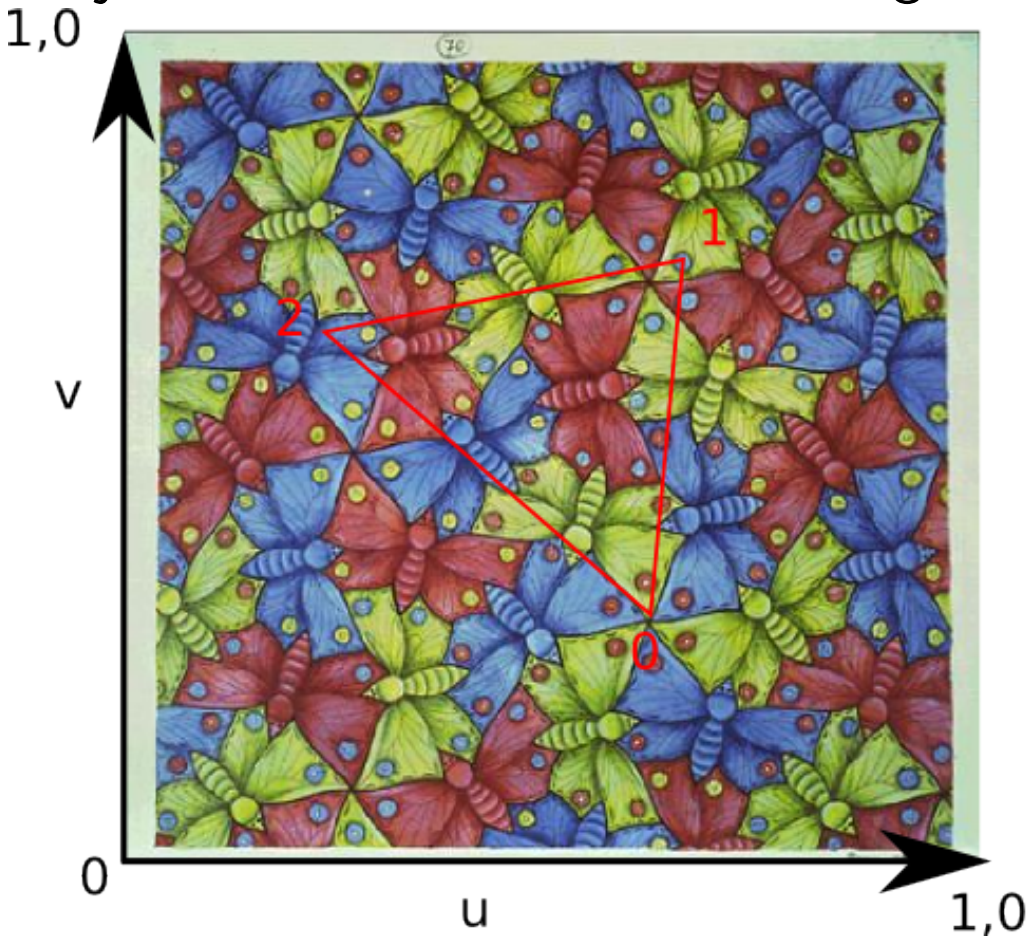
- rot, grün, blau werden als Fließkommazahl im Bereich $[0, 1.0]$ angegeben.
- alpha gibt die Deckkraft der Farbe an (Transparenz möglich). 1.0 ist maximale Deckung; 0.0 unsichtbar (komplett Transparent).

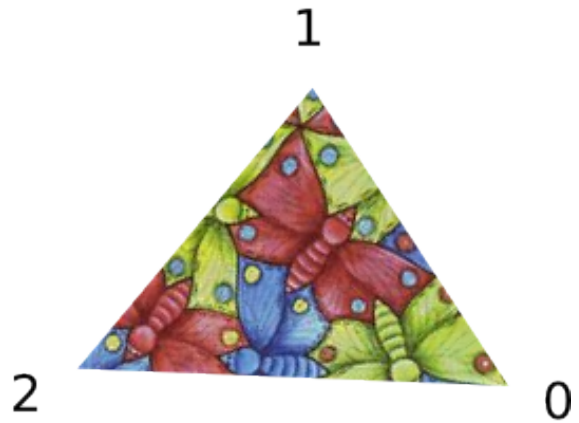
OpenGL-Befehle:

- `glColor4f(red, green, blue, alpha);`
  Setzt aktuelle Farbe; Die Farbe aller folgenden Pixel wird mit aktueller Farbe multipliziert.
- Eigene Farbe pro Punkt:
  `glEnableClientState(GL_COLOR_ARRAY);`
  `glColorPointer(4, GL_FLOAT, 0, colorBuffer);`

Texture Mapping

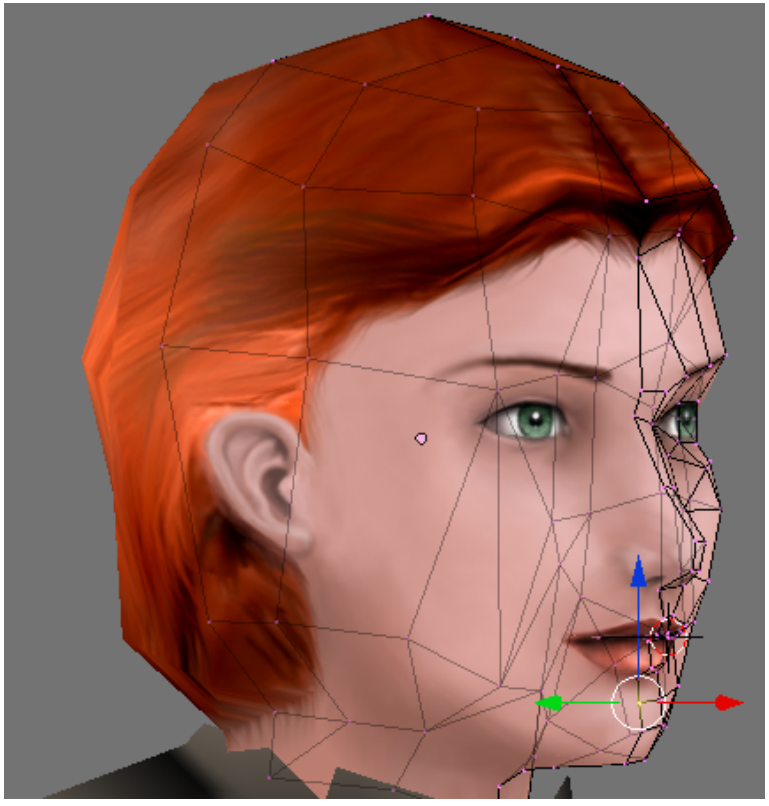
Projeziere ausschnitt aus Rechteckiger Textur auf ein Dreieck.

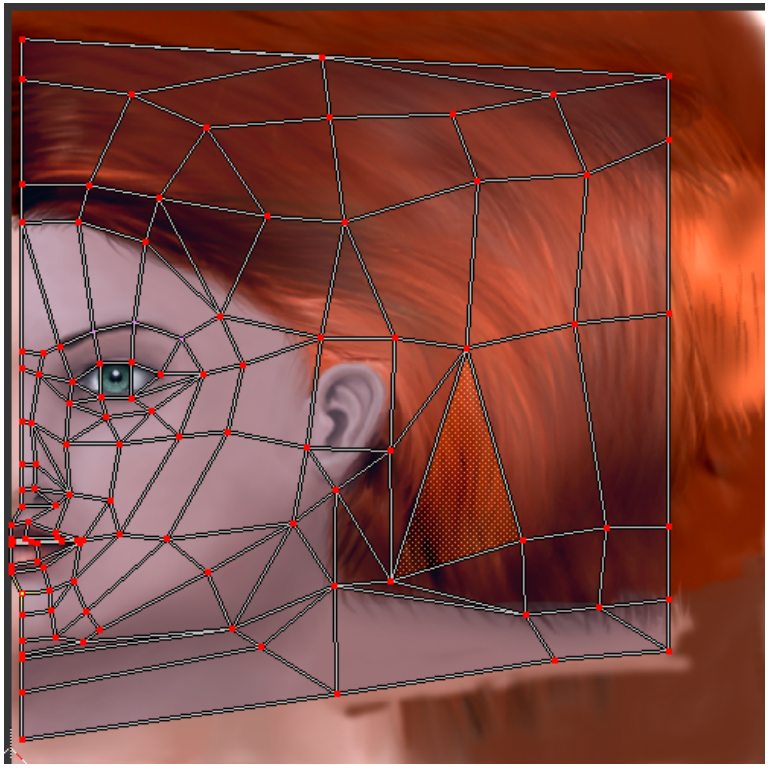




Die Koordinaten innerhalb der Textur werden mit Vektor (u, v) angegeben.

Texture Mapping – Beispiel





Textur laden - Randbedingungen

OpenGL verlangt folgendes bevor Texturen funktionieren:

- OpenGL schreibt vor, dass die Höhe und Breite der Textur eine Zweierpotenz ist (4, 8, 16, . . .).
- `GL_TEXTURE_MIN_FILTER` und `GL_TEXTURE_MAG_FILTER` müssen gesetzt werden.

Achtung: Der Android Software Emulator überprüft diese Dinge nicht und funktioniert trotzdem. Auf echten Handys läuft die Software dann aber nicht.

Textur laden

(Android Code)

```
/** Erzeugt OpenGL Textur aus einer Bitmap @return TexturID */
private int loadGlTexture(GL10 gl, Bitmap bitmap) {
    int[] tids = new int[1];
    gl.glEnable(GL10.GL_TEXTURE_2D);
    gl.glGenTextures(1, tids, 0);

    gl.glBindTexture(GL10.GL_TEXTURE_2D, tids[0]);
    /* check for power of 2 sizes */
    assert (bitmap.getWidth() & (bitmap.getWidth()-1)) == 0;
    assert (bitmap.getHeight() & (bitmap.getHeight()-1)) == 0;
    GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);
    gl.glTexParameterx(GL10.GL_TEXTURE_2D,
                       GL10.GL_TEXTURE_MIN_FILTER,
                       GL10.GL_LINEAR);
    gl.glTexParameterx(GL10.GL_TEXTURE_2D,
                       GL10.GL_TEXTURE_MAG_FILTER,
                       GL10.GL_LINEAR);
    return tids[0];
}
```

Mit Textur zeichnen

- Sicherstellen dass `glEnable(GL_TEXTURE_2D)` aktiviert ist.
- Zusätzlichen Puffer für Texturkoordinaten anlegen. Für jeden Punkt (Vertex) genau eine Texturkoordinate.
- Puffer mit Koordinaten vor dem Zeichnen angeben.

```
glEnableClientState(GL_TEXTURE_COORD_ARRAY);
glTexCoordPointer(2, GL_FLOAT, 0, coordBuffer);
```

- Textur auswählen („binden”): `glBindTexture(GL_TEXTURE_2D, texId)`
- wie gewohnt mit `glDrawElements(...)` zeichnen.

Typische Zeichenroutine

```
/* client State setzen (koennte man auch global fuers
 * ganze Programm machen */
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_TEXTURE_COORD_ARRAY);

glTexCoordPointer(2, GL_FLOAT, textureId);
glVertexPointer(3, GL_FLOAT, 0, vertexBuffer);
glDrawElements(GL_TRIANGLES, n_triangles_vertices,
                    GL_UNSIGNED_SHORT, indicesBuffer);
```

siehe auch: `drawEarth()`, `drawSphere()` im Beispiel.

Fehlerflag abfragen

```
private static String glErrorToString(int error) {
    switch (error) {
        case GL10.GL_NO_ERROR: return "no error";
        case GL10.GL_INVALID_ENUM: return "invalid enum";
        case GL10.GL_INVALID_OPERATION: return "invalid operation";
        case GL10.GL_INVALID_VALUE: return "invalid value";
        case GL10.GL_STACK_OVERFLOW: return "stack overflow";
        case GL10.GL_STACK_UNDERFLOW: return "stack underflow";
        case GL10.GL_OUT_OF_MEMORY: return "out of memory";
        default: return "unknown error";
    }
}
private static void checkForGlErrors(GL10 gl) {
    int error = gl.glGetError();
    if (error != GL10.GL_NO_ERROR) {
        throw new RuntimeException("OpenGL Error: " + glErrorToString(erro
    }
}
```

Jetzt einfach ab und zu `checkForGlErrors()` aufrufen.

Literatur


Khronos Group.
OpenGL ES reference pages.
http://www.khronos.org/opengles/sdk/docs/man/.


NeHe Productions.
OpenGL tutorials.
http://nehe.gamedev.net/.


Dave Shreiner, Mason Woo, and Jackie Neider.
OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL, Version 1.2.
Addison-Wesley Longman, Amsterdam, 6th edition, August 2007.