Proseminar Computergrafik: OpenGL

Marcel Heckel

23.05.2006

0. Inhalt

1. Allgemein
- Was ist OpenGL
- Geschichte

2. Etwas 3D-Mathematik
- Das Koordinatensystem
- Vektoren
- Matrizen

3. Grundlegendes zu OpenGL
- Arbeitsweise
- Vertices und Primitiven
- Bibliotheken
- Funktionsbezeichnungen und Datentypen
- GLUT-Beispiel

4. Zeichen
- Primitiven zeichnen
- Das ganze mit etwas Farbe
- Vertex-Behandlung
- Tests vor dem Schreiben in den Puffer
- Wichtige Funktionen

5. Licht
- Sinn von Licht
- Lichtarten
- Licht und Normalen
- Licht aktivieren

6. Texturen
- Sinn von Texturen
- Allgemein
- Texturkoordinaten und Wiederholung
- Filter
- Mip-Maps
- Texturen definieren

Was ist OpenGL?

- OpenGL steht für Open Graphics Library
- geräte- und betriebssystemunabhängige Bibliothek für qualitative 2D und 3D Grafiken und grafisches Rendering
- von Silicon Graphics Inc. (SGI) entwickelt
- Ursprünglich OpenGL für den Einsatz auf hochwertigen grafischen Workstations
- Heute weit verbreiteten Standard vieler Betriebssysteme
- von allen modernen Grafikkarten unterstützt
- zweithäufigste genutzte Software-Schnittstelle (nach Direct3D)

Geschichte

- Entstand aus IRIS GL (SGI)
- 1992 OpenGL-Standard von ARB (Architecture Review Board) definiert
- ARB Mitglieder (u.a.): Apple, ATI, Dell, IBM, Intel, NVidia, SGI, Sun
- Microsoft (Gründungsmitglied) verlies ARB 2003

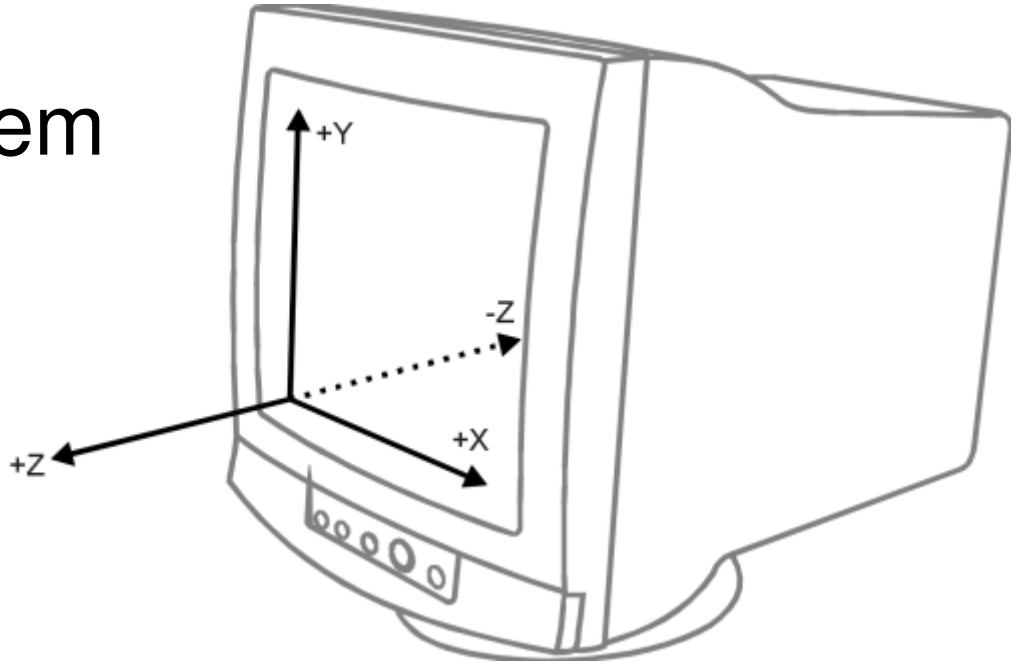
Geschichte

Versionsfolge:

- 1. Juli 1992: OpenGL 1.0
- 1997 OpenGL 1.1
- 16. März 1998: OpenGL 1.2
- 14. Oktober 1998: OpenGL 1.2.1
- 14. August 2001: OpenGL 1.3
- 24. Juli 2002: OpenGL 1.4
- 29. Juli 2003: OpenGL 1.5
- 7. September 2004: OpenGL 2.0

Das Koordinatensystem

- Kartesisches Koordinatensystem
- x-Achse: unterer Bildschirm-Rand
- y-Achse: linker Bildschirm-Rand
- in den Monitor blickt man entlang der negativen z-Achse



Vektoren

- Länge ein Vektors: $|v| = \sqrt{v.x^2 + v.y^2 + v.z^2}$
- Einheitsvektor: $v' = v / |v|$
- (Punkt-/)Skalarprodukt:

  $v \bullet u = v.x \cdot u.x + v.y \cdot u.y + v.z \cdot u.z$

  $= \cos a \cdot |v| \cdot |u|$
- Kreutzprodukt:

  $v = u \times w = (u.y \cdot w.z - u.z \cdot w.y ; u.z \cdot w.x - u.x \cdot w.z ; u.x \cdot w.y - u.y \cdot w.x)$

Matrizen

- Viele Berechnungen über 4x4-Mariezen (z.B. Position/Ausrichtung, Projektion, …)

a01	a05	a09	a13
a02	a06	a10	a14
a03	a07	a11	a15
a04	a08	a12	a16

- Grundzustand ist die Einheitsmatrix
- Nutzung als Position/Ausrichtung:
  (a01, a02, a03) – Links
  (a05, a06, a07) – Oben
  (a09, a10, a11) – Vorn
  (a13, a14, a15) – Position

Matrizen

Translation:

$$T = \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Skalierung:

$$S = \begin{pmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation:

$$R_X = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad R_Y = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad R_Z = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Matrizen

Multiplikation:

- 2 Matrix-Operation werden durch Matrix-Multiplikation verbunden
- Matrix-Multiplikation ist nicht kommutativ
- → Wichtig welche Reihenfolge:
  erst Rotation und dann Translation, oder umgekehrt.

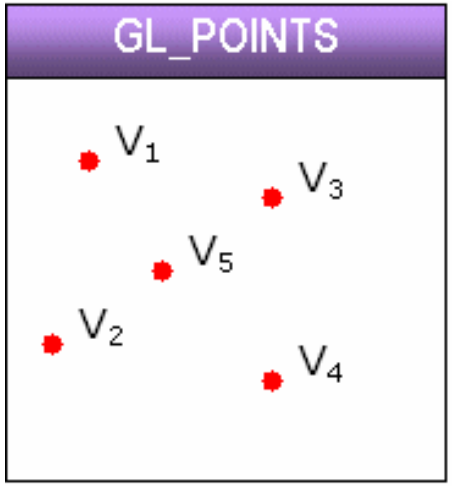
Arbeitsweise

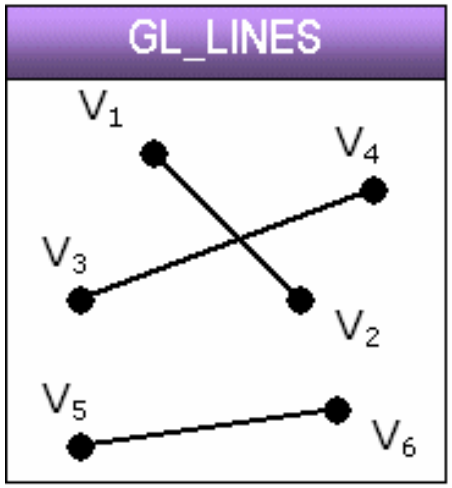
- schlankes hardwareunabhängiges Interface
- keine Unterstützung für Fenster bzw. Benutzereingaben
- keine Unterstützung für komplizierte 3D-Formen (Autos, Menschen, Bäume, …)
- Modell werde aus einfachen Primitiven wie Dreiecke, Linien und Punkte zusammengesetzt

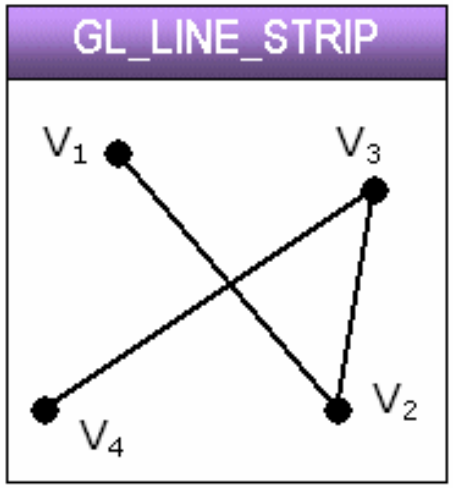
Arbeitsweise

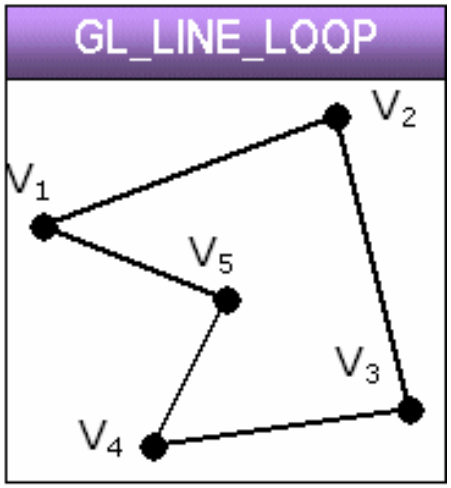
- OpenGL ist ein Zustandsautomat
- Einmal gesetzte Einstellungen beleiben erhalten und gelten für die folgenden Operationen

Vertices und Primitiven

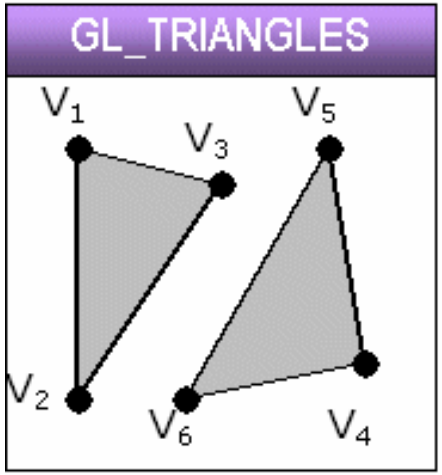


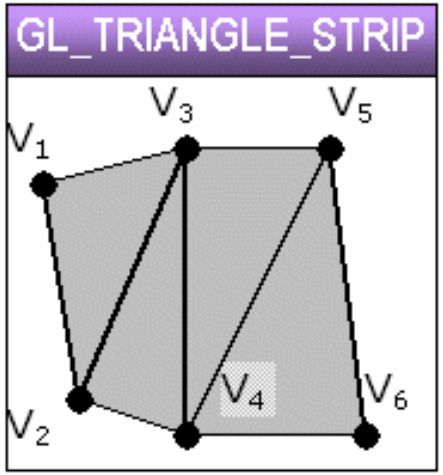


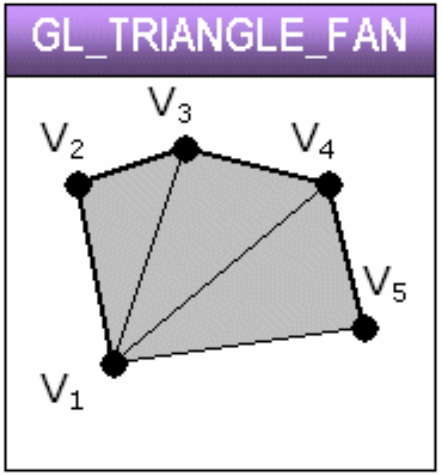


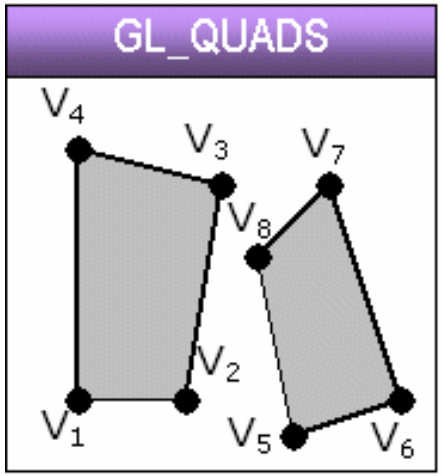


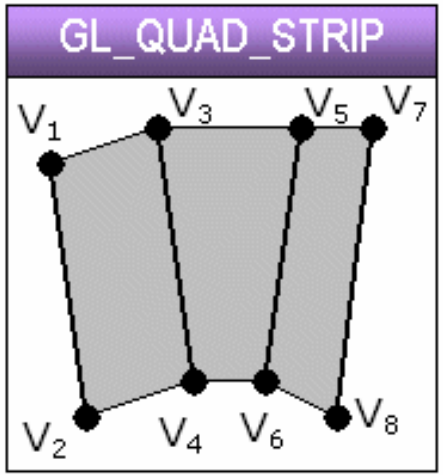
Vertices und Primitiven

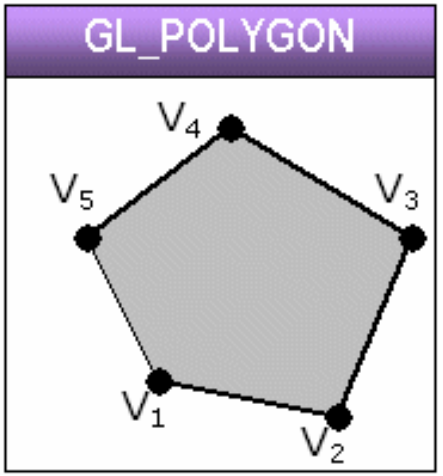






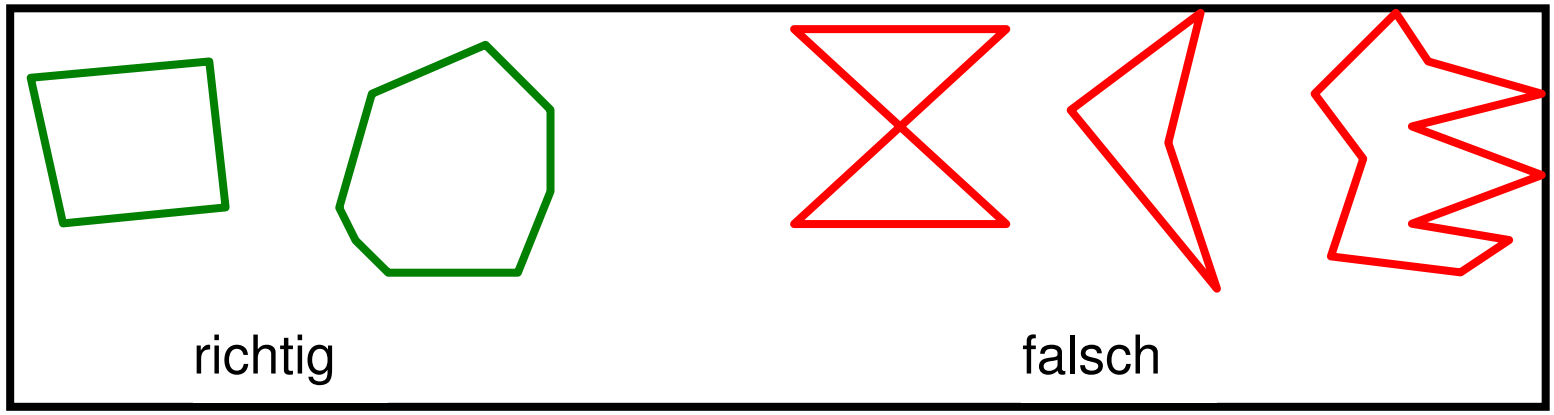






Vertices und Primitiven

- Vierecken und Polygonen müssen konvex (nach außen gewölbt) sein, sonst nicht definierte Ausgabe



Bibliotheken

- Standardbibliothek: opengl32.dll
- Alternativ: glew32.dll (mit Extrension-Unterstützung)
- glu.dll - OpenGL Utility Library (viele nützliche Standardfunktionen zum Zeichnen)
- glut.dll - OpenGL Utility Toolkit (Funktionalität zu Fenstererzeugung und Nutzereingaben)

Funktionsbezeichnungen und Datentypen

- Beginnen mit gl…/glu…/glut…
- Funktionen nicht überladen
  → verschiedene Funktionssuffixe für verschieden Datentypen
- Eigene Datentypbezeichnungen z.B.: GLint

Funktionsbezeichnungen und Datentypen

Fkt.-Suffix	Datentyp	GL-Typ	C-Typ	Fkt.-Suffix	Datentyp	GL-Typ	C-Typ
b	8-Bit Integer	GLbyte	char	d	64-Bit Gleitkomma	GLdouble , GLclampd	double
s	16-Bit Integer	GLshort	short	ub	8-Bit vorzeichenloser Int.	GLubyte , GLboolean	unsigned char
i	32-Bit Integer	GLint, GLsizei	int	us	16-Bit vorzeichenloser Int.	GLushort	unsigned short
f	32-Bit Gleitkomma	GLfloat, GLclampf	float	ui	32-Bit vorzeichenloser Int.	GLuint , GLenum, GLbitfield	unsigned int

GLUT - Beispiel


```
void init()
{ /* Initialisierung des Programms */ }
void renderScene(void)
{
    /* Szene zecihen */
    void glutSwapBuffers();
}
void changeSize(int w, int h)
{}
void keyFunction(unsigned char key, int x, int y)
{}
int main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(400,400);
    glutCreateWindow("kleines OpenGL Programm");
    glutDisplayFunc(renderScene);
    glutIdleFunc(renderScene);
    glutReshapeFunc(changeSize);
    glutKeyboardFunc(keyFunction);
    init();
    glutMainLoop();
    return 0;
}
```

Primitiven zeichnen

- Vertex-Angaben mit `glVertex*()`
- nur zwischen `glBegin(...)` und `glEnd()`
- Bei `glBegin(GLenum mode)` die gewünschte Primitive angeben

Primitiven zeichnen

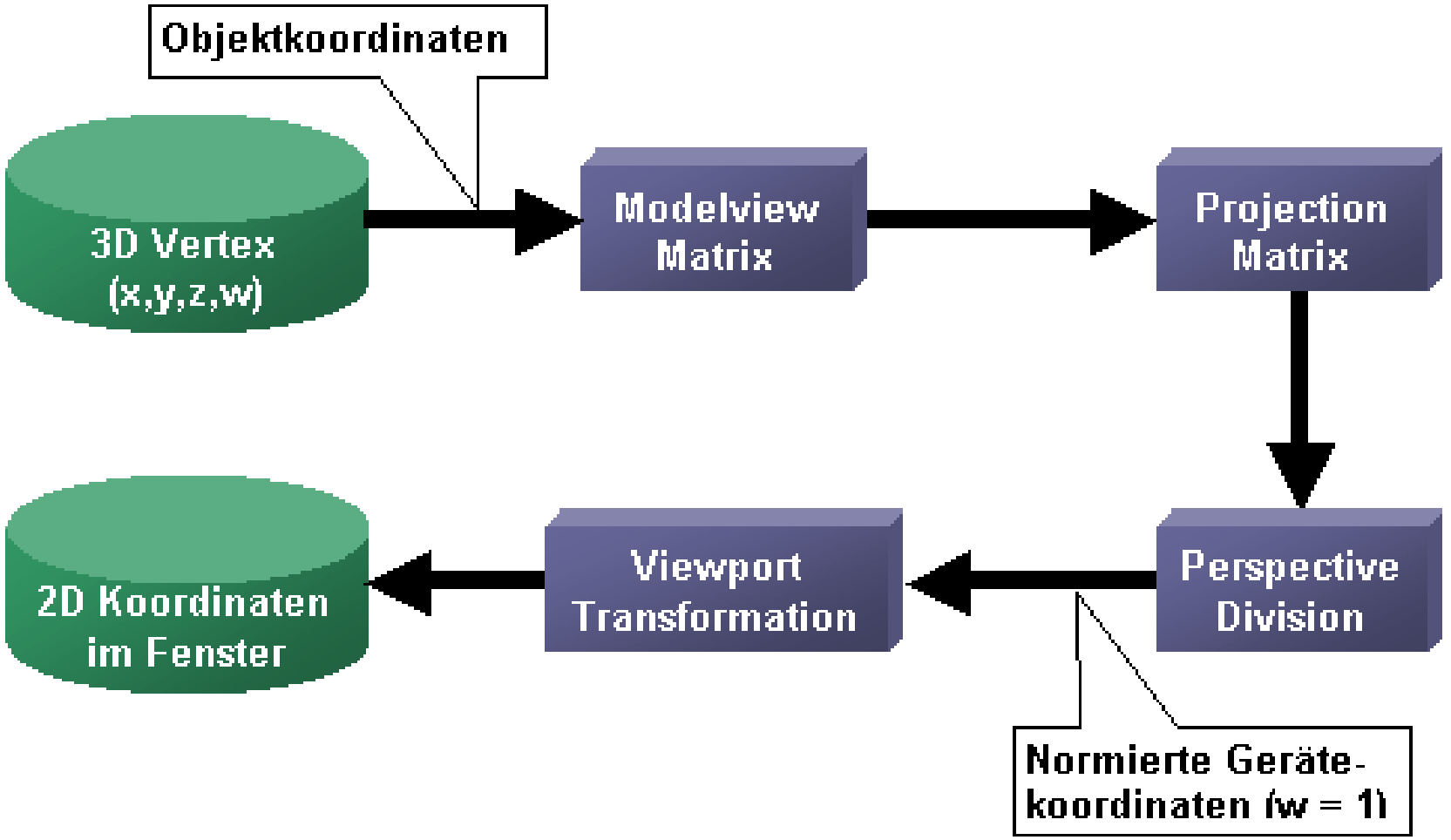

```
glBegin(GL_TRIANGLES);
   glVertex3d(0.0,0.5,0.0);
   glVertex3d(0.0,0.0,0.0);
   glVertex3d(0.5,0.5,0.0);

   glVertex3d(0.5,0.0,0.0);
   glVertex3d(0.0,-0.5,0.0);
   glVertex3d(0.5,-0.5,0.0);
glEnd();

glBegin(GL_QUADS);
   glVertex3d(-0.9,0.9,0.0);
   glVertex3d(-0.9,-0.9,0.0);
   glVertex3d(-0.1,-0.9,0.0);
   glVertex3d(-0.1,0.9,0.0);
glEnd();
```

Das ganze mit etwas Farbe

- Verschieden Farben für jedes Vertex möglich → Interpolation zw. den Farben jedes Vertices
- 4 Farbanteile- RGBA: A- Alphawert

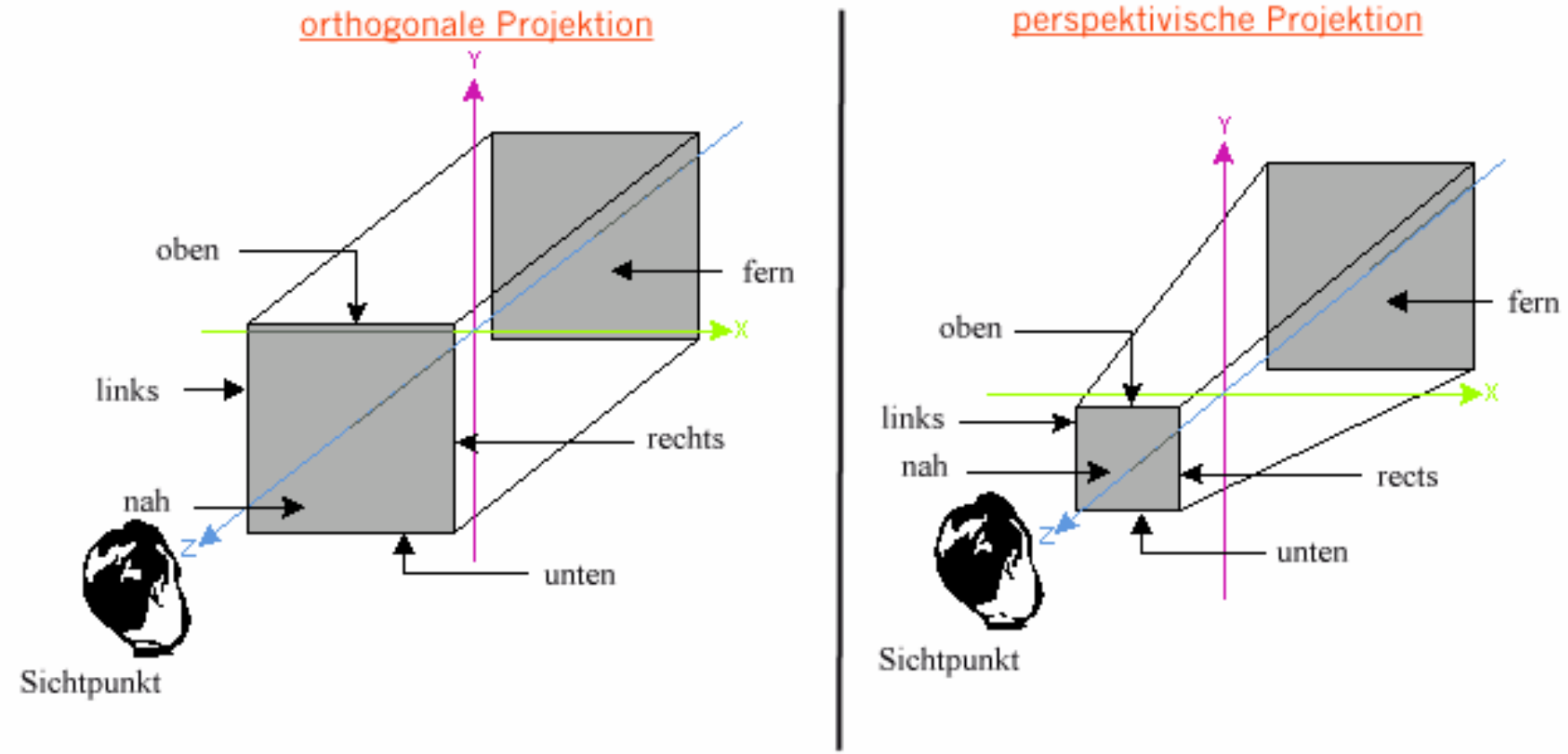
```
glBegin(GL_TRIANGLES);
  glColor3f(1.0f,0.0f,0.0f);
  glVertex3f( 0.0f, 1.0f, 0.0f);
  glColor3f(0.0f,1.0f,0.0f);
  glVertex3f(-1.0f,-1.0f, 1.0f);
  glColor3f(0.0f,0.0f,1.0f);
  glVertex3f( 1.0f,-1.0f, 1.0f);
glEnd();
```

Vertex-Behandlung



4. Zeichen → Vertex-Behandlung

Projektion



Tests vor dem Schreiben

- **Depth-Test:** wichtigster Test: Prüfung ob neues Pixel eine kleineren Tiefenwert hat als das alte
- **Alpha-Test:** Prüfung ob der Alfawert einer Bedingung entspricht (>,<,= Referenzwert)
- **Stencil-Test:** Bitmap welches die Bereiche kennzeichnet, in die nicht gezeichnet werden soll markiert.

Wichtige Funktionen

- `glMatrixMode();`
- `glLoadMatrix(); glMultMatrix();`
- `glPushMatrix(); glPopMatrix();`
- `glTranslate(); glRotate(); glScalar(); gluLookAt();`
- `glFrustum(); glOrtho(); gluPerspective();`
- `glViewport();`

Sinn von Licht

- 3D-Szenen wirken wesentlich realistischer
- Szenen in verschiedenen Farben darstellen ohne die ganzen Eigenschaften neu zu setzten
  z.B.: Tageslicht / Sonnenuntergang

Weiteres:

- OpenGL unterstützt bis zu 8 Lichtquellen
- Abstrahlwinkel und Richtung können auch definiert werden

Lichtarten

- *ambient (umgebendes) Licht*:
  allg. Lichtverhältnissen
- *diffuse (zerstreutes/ausbreitendes) Licht*:
  matte Oberflächen
- *specular (reflektiertes) Licht*:
  glänzenden Oberflächen

Normalen

- Oberflächen-Richtung muss für jede Primitive/Normale angegeben werden
- Sollten Länge 1 haben
- Beleuchtung der Fläche in Anteilen, wie die Normal zur Lichtquelle steht
- Gewölbte Flächen: Normalen mehr der Wölbung entsprechen lassen, nicht genau senkrecht zur Teilfläche

Licht aktivieren

- Licht ist standardmäßig ausgeschaltet:
```
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
```
- Einstellungen können mit `glLightf{v}` vorgenommen werden: Farbe, Position, Abstrahlwinkel, Abschwächung.

Sinn von Texturen

- Objekte mit vielen kleine Details währen sehr aufwendig zu zeichnen
- → Lösung: Über eine einfache Fläche ein Bild, das die Details zeigt, legen
- Ist wesentlich schneller und sieht (meist) auch besser aus.

Allgemein

- Arrays aus RGB(A)-Werten
- Bis OpenGL 1.5: Texture-Maße mussten 2er-Potenz sein (2^n) → seit 2.0 nicht mehr
- mögliche Dimensionen: 1D, 2D und 3D!!

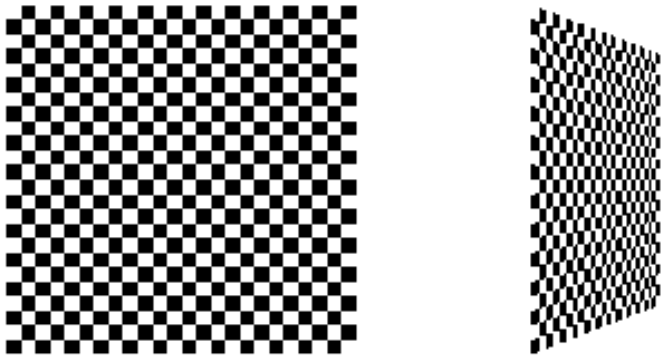
Texturkoordinaten und Wiederholung

- Angabe mit `glTexCoord()`
  2D: (0.0, 0.0) linke untere Ecke
  (1.0, 1.0) rechte obere Ecke
- Bezeichnung: s, t, r (Breite, Höhe, Tiefe)
- Bei Werten außerhalb [0.0; 1.0]:
  Wiederholung der Textur (std.)
  oder Streckung der letzten Reihe der Textur

Texturkoordinaten und Wiederholung

- `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_S_WRAP, *** );`
- `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_T_WRAP, *** );`

`*** = GL_REPEAT`



`*** = GL_CLAMP`

Filter

- Magnifikation: starke Vergrößerung
- Minifikation: starke Verkleinerung

Allgemein 2 Filtertypen:

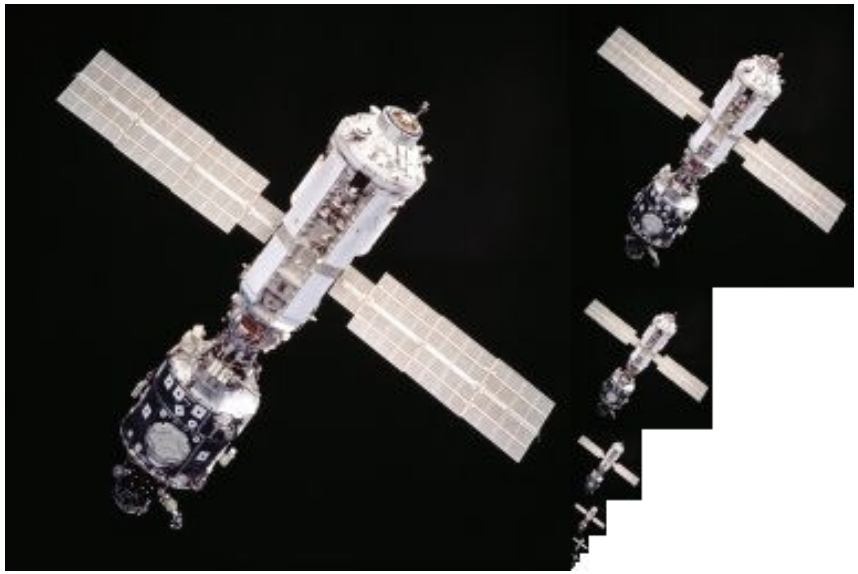
`GL_NEAREST`: das Pixel was am ehesten passt (große Pixel bei Vergrößerung)

`GL_LINEAR`: Mittelwert aus dem umgebenden Pixel (langsamer, unscharf bei Vergrößerung)

→ Verkleinerung: ähnliche Ergebnisse beider Verfahren; Problem: „blinkende“ Texturen

Mip-Maps

- Mip = „multum in parvo“ - „viel auf kleinem Platz“
- mehrere verkleinerte Versionen vom Originalbild
- Verhindert „blinkende“ Texturen.
- schneller



Texturen definieren

- `glEnable(GL_TEXTURE_2D);`
- `glTexImage2D` – normal
- `gluBuild2DMipmaps` – autom. Mipmap Erzeugung

→Aktuelle Einstellungen von `glTexParameter` werden für die Textur übernommen

Quellen

- Jackie Neider, Tom Davis, Mason Woo: „*OpenGL Programming Guide*“, Release 1, 1994
- http://nehe.gamedev.net
- http://www.lighthouse3d.com/opengl/glut/
- http://graphics.cs.uni-sb.de/Courses/ws9900/cg-seminar/Ausarbeitung/Philipp.Walter/
- http://www.opengl.org
- http://de.wikipedia.org