LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Prof. Dr. Andreas Butz | Prof. Dr. Ing. Axel Hoppe

Dipl.-Medieninf. Dominikus Baur
Dipl.-Medieninf. Sebastian Boring

Übung: Computergrafik 1

Licht, Material, Blending
Texturen

Licht

Licht

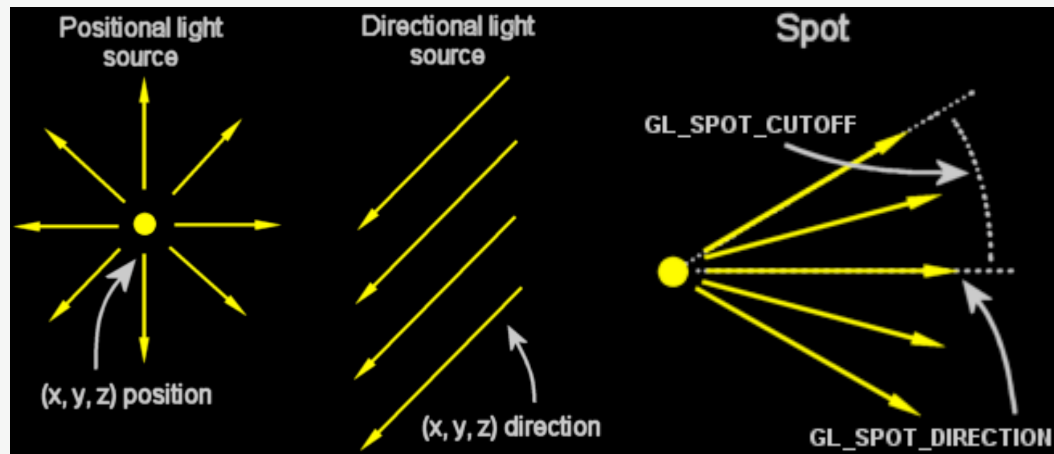
- Bevor Texturen oder Materialien auf Objekten angezeigt werden können muss eine Lichtquelle vorhanden sein
- Das Aussehen eines Objekts hängt immer von der Beleuchtung der jeweiligen Umgebung ab

- Lichtquellen haben in OpenGL zwei Haupteigenschaften:
  - Typ der Ausbreitung (d.h. Spotlight, Ambient, etc)
  - Lichtfarbe in verschiedenen Ausprägungen

- Lichtquellen müssen nur einmal zu Beginn initialisiert werden
- Primitive brauchen Normalen (s.u.) damit die Berechnung funktioniert

Ausbreitung

- Lichtquellen haben verschiedene Ausbreitungstypen:
  - Punktlichtquelle: nur Position (strahlt in alle Richtungen gleichmäßig)
  - Gerichtetes Licht: nur Vektor (unendlich weit entfernte Lichtquelle (z.B. Sonne)
  - Spotlight: Position, Richtungsvektor, Öffnungswinkel und Intensitätsabfall (Punktlichtquelle mit bestimmtem Öffnunskegel)



(Quelle: http://jerome.jouvie.free.fr/OpenGl/Tutorials/Tutorial13.php)

Lichtfarbe und -eigenschaften

- ambient: Ungerichtetes Umgebungslicht
- diffuse: Gerichtetes Licht, das in alle Richtungen reflektiert wird
- specular: Gerichtetes Licht, das nur in eine Richtung reflektiert wird

Eigenschaften von Lichtquellen:

- Position
- Richtungsvektor (für Spotlights)
- Öffnungswinkel (für Spotlights)
- Ambiente Farbe
- Diffuse Farbe
- Spiegelungsfarbe
- Intensitätsabfall
- Radialer Intensitätsabfall (für Spotlights)

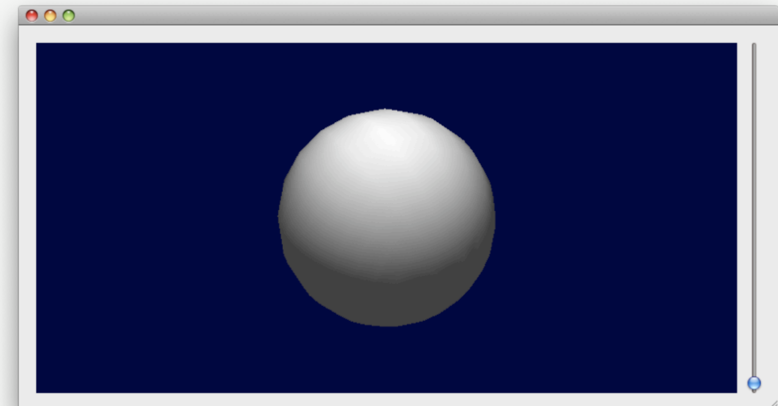
- `glEnable(GL_LIGHTING)` Aktivieren des OpenGL Lichts
- `glEnable(GL_LIGHTi)` Aktivieren der Lichtquelle *i*
- `glLightf(light, pname, param)` Anpassen der Eigenschaften einer Lichtquelle
  - `light` Lichtnummer (GL_LIGHT1, ...)
  - `pname` Parametername (GL_SPOT_CUTOFF, GL_LINEAR_ATTENUATION, etc)
  - `param` Neuer Wert
- `glLightfv(light, pname, *param)` Anpassen der Eigenschaften einer Lichtquelle mit Vektor
  - `light` Lichtnummer (GL_LIGHT1, ...)
  - `pname` Parametername (GL_AMBIENT, GL_DIFFUSE, GL_POSITION, etc.)
  - `*param` Pointer auf einen Vektor mit neuen Werten

(Quelle: http://www.opengl.org)

gltest.cpp

```
void GlTest::initLight(){
    GLfloat LightAmbient[] = {0.8f, 0.8f, 0.8f, 1.0f};
    GLfloat LightDiffuse[] = {1.0f, 1.0f, 1.0f, 1.0f};
    GLfloat LightPosition[]= { 0.0f, 4.0f, -5.0f, 1.0f };

    glEnable(GL_LIGHTING);

    glLightfv(GL_LIGHT1, GL_AMBIENT, LightAmbient);
    glLightfv(GL_LIGHT1, GL_DIFFUSE, LightDiffuse);
    glLightfv(GL_LIGHT1, GL_POSITION, LightPosition);
    glEnable(GL_LIGHT1);

}
```

```
glTranslatef(0, 0, -10);
glRotatef(rotaY, 0, 1, 0);

glPushMatrix();
GLUquadricObj* quad = gluNewQuadric();
gluSphere(quad, 2.5f, 20, 20);
```



Licht in OpenGL

- nur ambient light:

- nur diffuse light:

Lichteigenschaften

Parameter	Default	Bedeutung
`GL_AMBIENT`	(0.0, 0.0, 0.0, 1.0)	Ambiente Farbe des Lichts
`GL_DIFFUSE`	(1.0, 1.0, 1.0, 1.0)	Diffuse Farbe des Lichts
`GL_SPECULAR`	(1.0, 1.0, 1.0, 1.0)	Spiegelungsfarbe
`GL_POSITION`	(0.0, 0.0, 1.0, 1.0)	Position
`GL_SPOT_DIRECTION`	(0.0, 0.0, –1.0, 0.0)	Richtung (Spotlight)
`GL_SPOT_EXPONENT`	0.0	Radiale Intensität (Spotlight)

(Quelle: http://www.opengl.org)

Lichteigenschaften

Parameter	Default	Bedeutung
GL_SPOT_CUTOFF	180.0	Halber Öffnungswinkel (Spotlight)
GL_CONSTANT_ATTENUATION	1.0	Konstanter Abschwächungsfaktor
GL_LINEAR_ATTENUATION	0.0	Linearer Abschwächungsfaktor
GL_QUADRATIC_ATTENUATION	0.0	Quadratischer Abschwächungsfaktor

(Quelle: http://www.opengl.org)

Directional und Spotlight

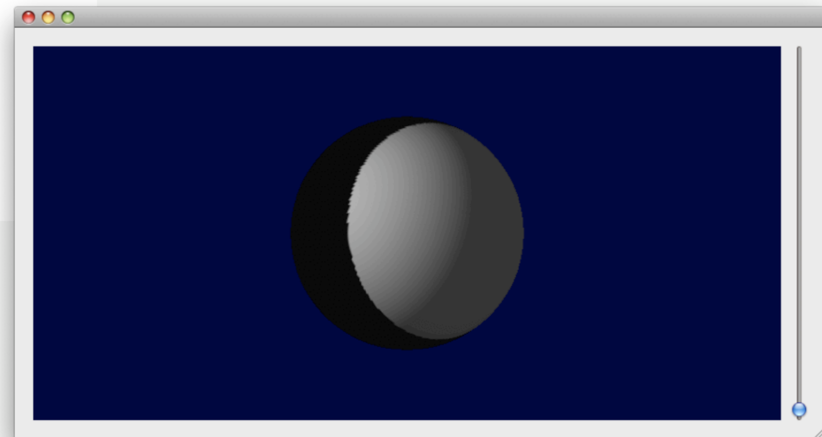
- Die Lichtposition ist ein vierdimensionaler Vektor (x,y,z,w)
- Für w = 1 (Default) wird eine Punktlichtquelle erzeugt
- Falls w = 0 wird gerichtetes Licht mit dem Positionsvektor als Richtung erzeugt
- Für ein Spotlight muss w = 1 sein, mit GL_SPOT_DIRECTION wird die Richtung bestimmt

(Quelle: http://www.opengl.org)

Spotlight in OpenGL

gltest.cpp

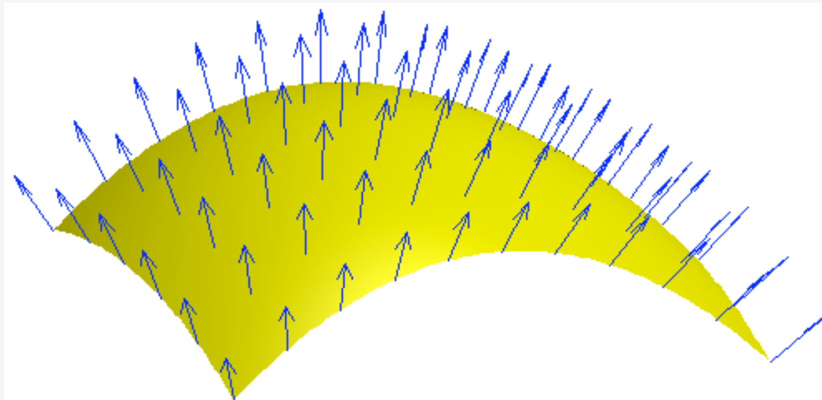
```
void GlTest::initLight(){
    GLfloat LightAmbient[] = {0.8f, 0.8f, 0.8f, 1.0f};
    GLfloat LightDiffuse[] = {1.0f, 1.0f, 1.0f, 1.0f};
    GLfloat LightPosition[]= { -5.0f, 2.0f, -5.0f, 1.0f };

    GLfloat spotDirection[] = {1.0f, 0.0f, 1.0f};

    glEnable(GL_LIGHTING);

    glLightfv(GL_LIGHT1, GL_AMBIENT, LightAmbient);
    glLightfv(GL_LIGHT1, GL_DIFFUSE, LightDiffuse);
    glLightfv(GL_LIGHT1, GL_POSITION, LightPosition);
    glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION, spotDirection);
    glLightf(GL_LIGHT1, GL_SPOT_CUTOFF, 80);
    glLightf(GL_LIGHT1, GL_SPOT_EXPONENT, 0.2f);
    glEnable(GL_LIGHT1);

}
```



Objekteigenschaften

Normalen

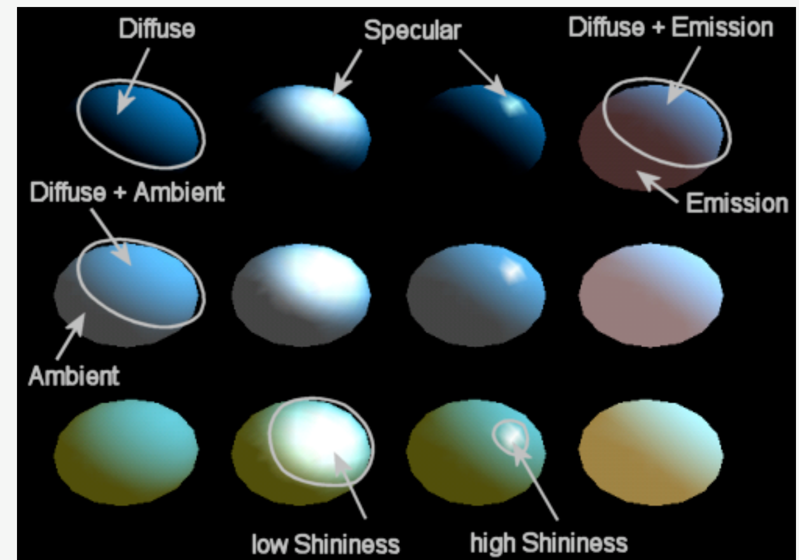
- Damit die Lichtberechnung vernünftig funktioniert sollte jedes Polygon in der Szene eine Normale haben
- Normalen sind ebenfalls ein angenehmerer Weg Vorder- und Rückseite eines Objekts zu definieren
- `glNormal3f/4fv` verändern die aktuelle Normale (Voreinstellung (0,0,1))
- Mit glu erzeugte Objekte (z.B. gluSphere) haben bereits passende Normalen
- Ebenfalls wichtig für Texturierung



(Quellen: http://www.opengl.org http://en.wikipedia.org/wiki/Surface_normal)

Lichtfarbe und –eigenschaften

- ambient: Grundfarbe des Objekts
- emissive: Farbe in der das Objekt (von sich aus) leuchtet (Glühen)
- diffuse: Licht, das in alle Richtung reflektiert wird
- specular: Licht, das nur in eine Richtung reflektiert wird



Materialien in OpenGL

- Setzen von Materialeigenschaften mit
- `glMaterialf(face, pname, param)`
- `glMaterialfv(face, pname, *param)`
  - `face` GL_FRONT, GL_BACK, GL_FRONT_AND_BACK
  - `pname` Parametername (GL_SHININESS, GL_AMBIENT, GL_DIFFUSE, etc)
  - `param` Neuer Wert
  - `*param` Pointer auf einen Vektor mit neuen Werten

(Quelle: http://www.opengl.org)

Materialeigenschaften

Parameter	Default	Bedeutung
`GL_AMBIENT`	(0.2, 0.2, 0.2, 1.0)	Ambiente Farbe (allgemeine Farbe)
`GL_DIFFUSE`	(0.8, 0.8, 0.8, 1.0)	Diffuse Farbe (Farbe durch Licht)
`GL_SPECULAR`	(0.0, 0.0, 0.0, 1.0)	Spiegelungsfarbe (abhängig von Pos.)
`GL_SHININESS`	0.0	Glanzpunktstärke (je höher, desto heller)
`GL_EMISSION`	(0.0, 0.0, 0.0, 1.0)	Farbe des ausgesandten Lichts
`GL_COLOR_INDEXES`	(0, 1, 1)	Ambient-, Diffus-, Spiegelungsindex

(Quelle: http://www.opengl.org)

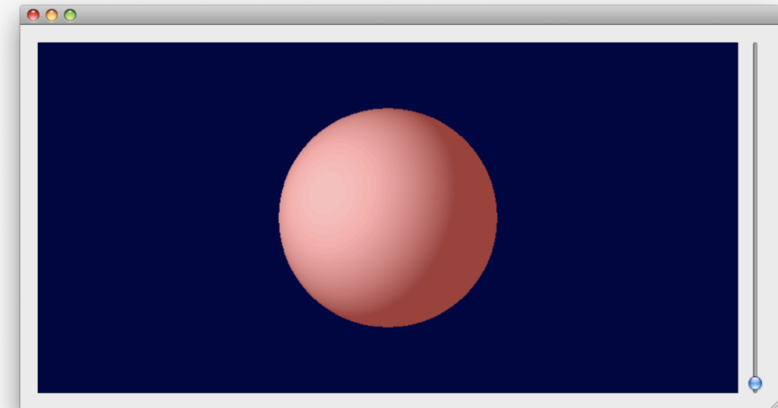
Beispielmaterialien

Material	r_{ar}, r_{ag}, r_{ab}	r_{dr}, r_{dg}, r_{db}	r_{sr}, r_{sg}, r_{sb}	n
Plastik	0.0, 0.0, 0.0	0.01, 0.01, 0.01	0.50, 0.50, 0.50	32
Messing	0.33, 0.22, 0.03	0.78, 0.57, 0.11	0.99, 0.94, 0.81	28
Bronze	0.21, 0.13, 0.05	0.71, 0.43, 0.18	0.39, 0.27, 0.17	26
Kupfer	0.19, 0.07, 0.02	0.70, 0.27, 0.08	0.26, 0.14, 0.09	13
Gold	0.25, 0.20, 0.07	0.75, 0.61, 0.23	0.63, 0.56, 0.37	51
Silber	0.19, 0.19, 0.19	0.51, 0.51, 0.51	0.51, 0.51, 0.51	51

Materialien in OpenGL

gltest.cpp

```
GLfloat amcol[] = {0.59f, 0.19f, 0.19f};
GLfloat specCol[] = {0.51f, 0.51f, 0.51f};
glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, amcol);
glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, specCol);
glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, specCol);
glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 0.51f);

GLUquadricObj* quad = gluNewQuadric();
gluSphere(quad, 2.5f, 200, 200);
```



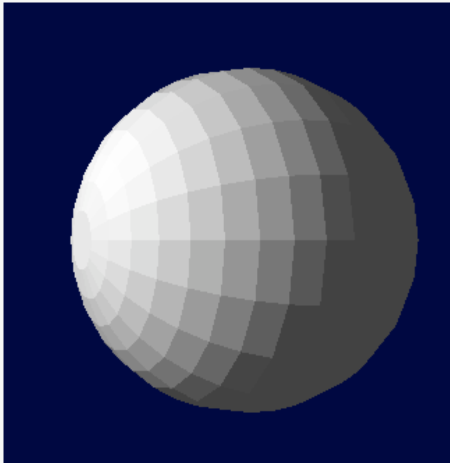
Farbmischung in OpenGL

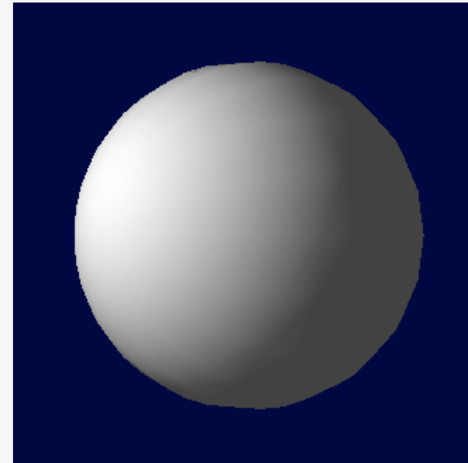
- Sowohl Licht als auch Materialien haben verschiedene (Ambient, Diffuse, etc) Farben. Wie ergibt sich daraus die Farbe des Polygons?
- Die Materialfarbe bestimmt, welche Teile des ankommenden Lichts reflektiert werden
- Beispiel:
  - Eine vollständig rote Kugel (R=1,G=B=0) reflektiert alles ankommende rote Licht
  - Bei einer weißen Lichtquelle sind die Farben gleichmäßig verteilt, sodass die Kugel rot erscheint
  - Bei einer roten Lichtquelle sieht die Kugel ebenfalls rot aus
  - Bei einer grünen Lichtquelle wird die Kugel schwarz, da kein rotes Licht reflektiert werden kann

(Quelle: http://glprogramming.com/red/chapter05.html)

Shading

- Die Berechnung der Farbe einer Oberfläche heißt Shading
- Je nach Materialeigenschaften, Lichtquellen und Einstellungen ergeben sich unterschiedliche Farben
- `glShadeModel` bestimmt den angewendeten Shading-Modus:
  - `GL_FLAT` Flat shading (Shading pro Polygon)
  - `GL_SMOOTH` Gouraud shading (Shading pro Pixel) (Voreinstellung)

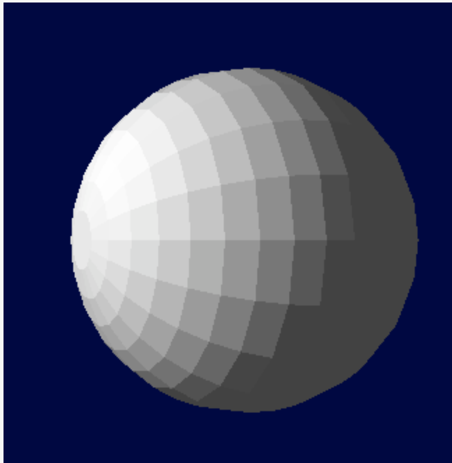




(Quelle: www.opengl.org)

Flat Shading

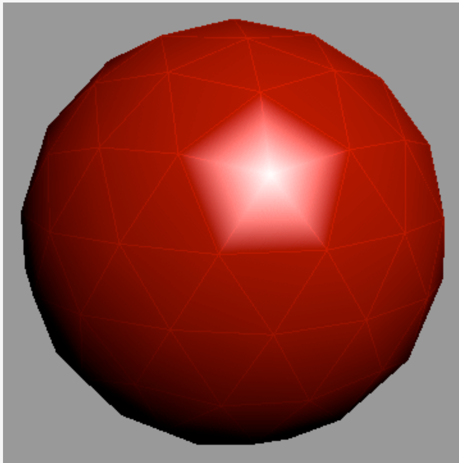
- Beim Flat oder Constant Shading wird jedem Polygon eine konstante Farbe zugewiesen
- Die Berechnung dieser Farbe basiert auf der Oberflächennormalen und Abstand, Farbe und Richtung der Lichtquelle(n)
- Funktioniert gut bei ebenen Oberflächen (Würfel), schlecht bei gekrümmten (Kugel)
- sehr schnell, aber in Zeiten von hardwarebeschleunigter Grafik eigentlich nicht mehr nötig

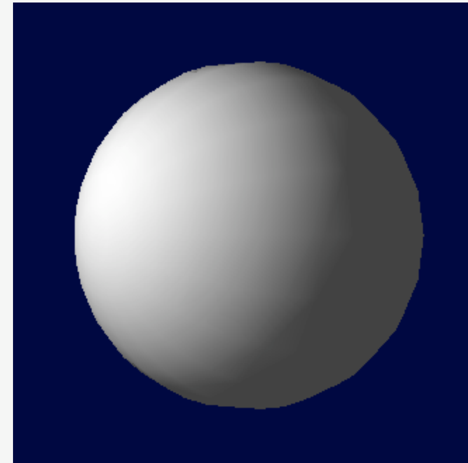


(Quelle: http://de.wikipedia.org/wiki/Flat_Shading)

Gouraud Shading

- Beim Gouraud Shading wird zuerst für jeden Vertex der Fläche die Farbe berechnet
- Die Farbe eines Pixels auf der Oberfläche ergibt sich dann durch Interpolation
- Generell besser als Flat Shading, versagt aber bei Glanzpunkten
- Verbesserung: Phong Shading
  - pixelweise Interpolation des Normalenvektors
  - in OpenGL nur per Shading Language



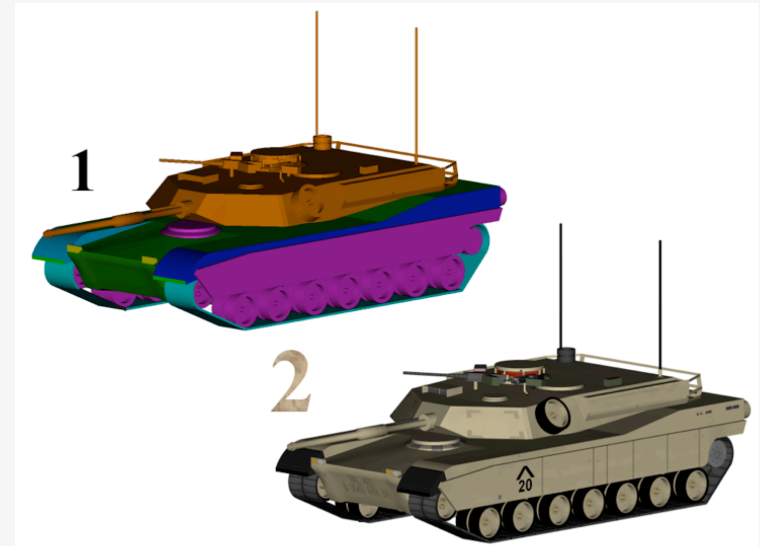


(Quelle: http://en.wikipedia.org/wiki/Gouraud_shading)

Texturierung

Texturierung

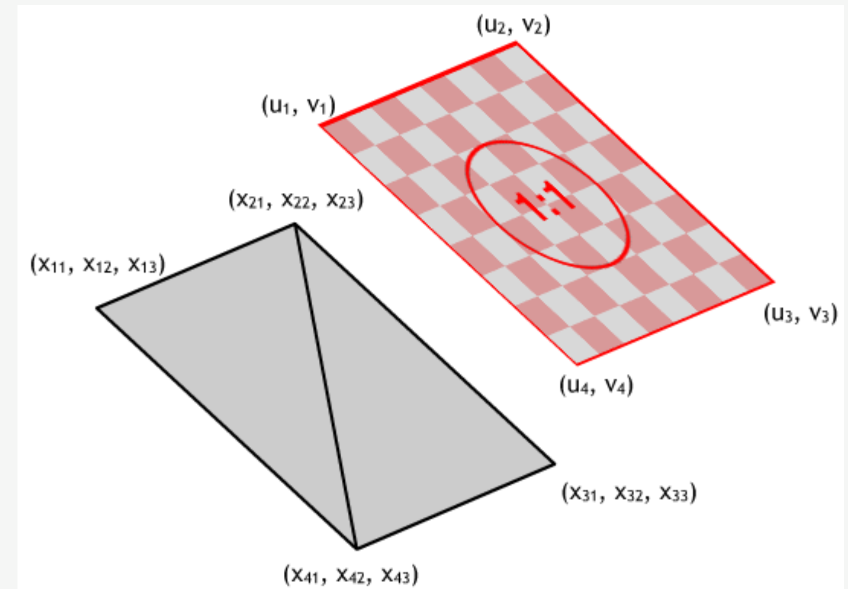
- Eine realistische Nachbildung von Oberflächen ist nicht nur rechen- sondern auch arbeitsaufwändig
- Meistens reicht es dass Oberflächen nur so aussehen “als ob” und nicht vollständig modelliert sind
- Texturierung (Texture Mapping) ist ein Prozess bei dem zweidimensionale Bitmapbilder über dreidimensionale Oberflächen gezogen werden



(Quelle: http://en.wikipedia.org/wiki/Texture_mapping)

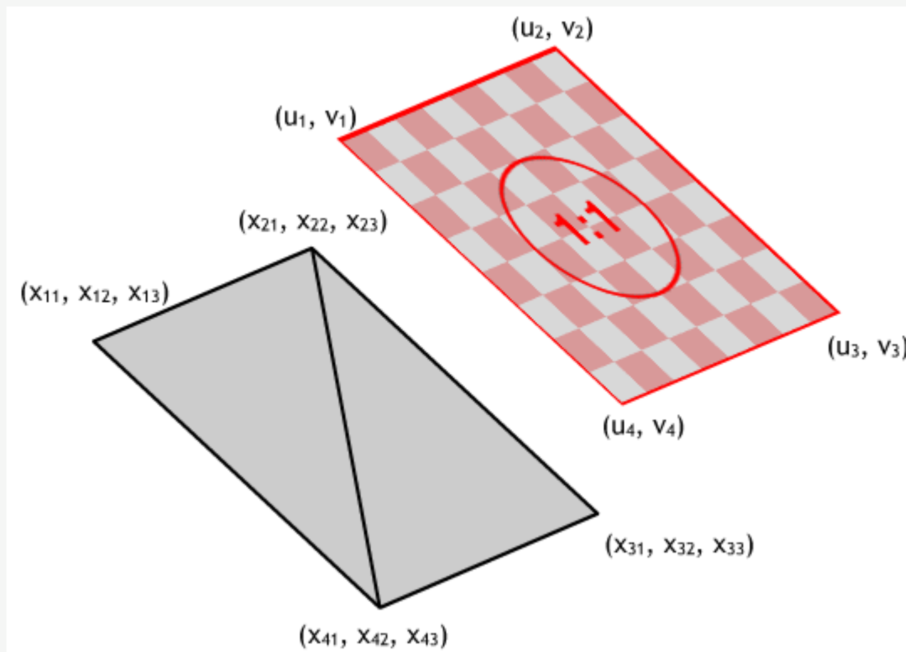
Texturkoordinaten

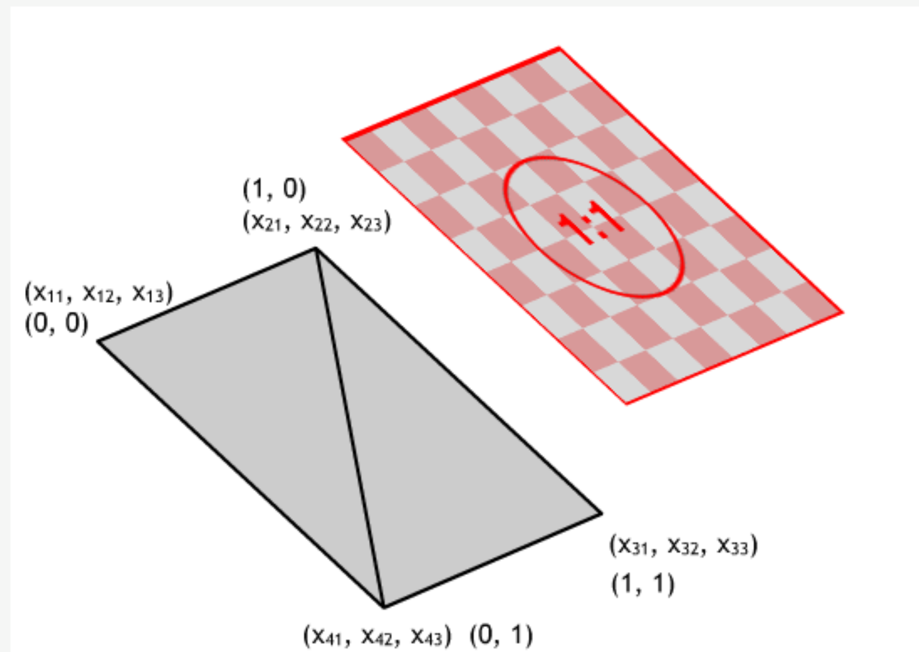
- Wie bringt man zweidimensionale Bilder auf dreidimensionale Körper?
- In OpenGL hat jeder Punkt eines Polygons zusätzliche 2D-Texturkoordinaten (u, v)
- Die Textur wird zwischen diesen Punkten interpoliert
- Texturkoordinaten liegen im Bereich zwischen 0 und 1 auch für nicht-quadratische Texturen



(Quelle: http://www.medien.ifi.lmu.de/lehre/ss08/3dp/ProgPrakt_3D_Szenengraph_Texturen.pdf)

Texturkoordinaten





(Quelle: http://www.medien.ifi.lmu.de/lehre/ss08/3dp/ProgPrakt_3D_Szenengraph_Texturen.pdf)

Texturen in OpenGL

- Setzen von Texturkoordinaten mit `glTexCoord2d/f`
- Kann auch in glBegin...glEnd Blöcken angewendet werden

```
glBegin(GL_QUADS);
    glTexCoord2f(0, 0);
    glVertex3f(2, 2, 0);
    glTexCoord2f(1, 0);
    glVertex3f(-2, 2, 0);
    glTexCoord2f(1, 1);
    glVertex3f(-2, -2, 0);
    glTexCoord2f(0, 1);
    glVertex3f(2, -2, 0);
glEnd();
```

(Quelle: http://www.opengl.org)

Texturen in OpenGL

- Aktivierung des Texture Mappings in OpenGL mit `glEnable(GL_TEXTURE_2D)` (Texture Mapping kann genauso wie Licht beliebig an- und abgestellt werden)
- `glGenTextures(num, GLuint* target)` generiert `num` neue Texturen und speichert das Ergebnis in `*target`
- `glBindTexture(target, texture)` aktiviert eine bestimmte Textur
  - target meistens GL_TEXTURE_2D
  - texture Nummer der Textur

(Quelle: http://www.opengl.org)

Texturen in OpenGL

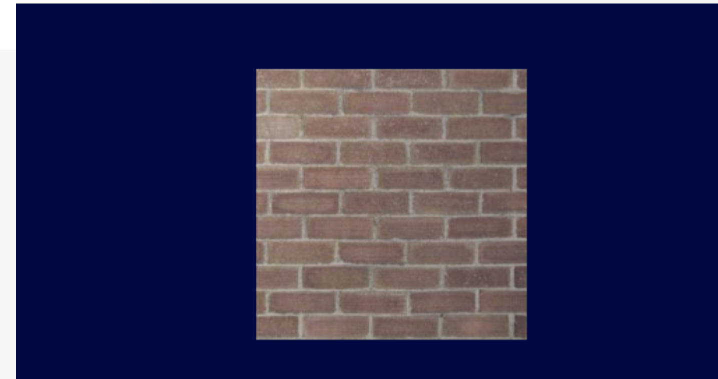
- Das Laden einer Textur erfolgt mit
- `glTexImage2D (target, level, internalformat, width, height, border, format, type, *pixels)`
  - `target` meist GL_TEXTURE_2D
  - `level` Level-Of-Detail (s.u.). Meist 0
  - `internalformat` Anzahl der Farbkanäle (1 - 4)
  - `width, height` Bilddimensionen
  - `border` Rahmengröße (0 - 1)
  - `format` Pixeldatenformat (bei QImage GL_RGBA)
  - `pixels` Pointer zu den Bilddaten
- Die Breite und Höhe einer Textur muss eine Zweierpotenz sein, d.h. 128 x 128, 128 x 256, 256 x 256, 512 x 512, ... (Maximalgröße hängt von Grafikhardware ab)

(Quelle: http://www.opengl.org)

Texturen in OpenGL

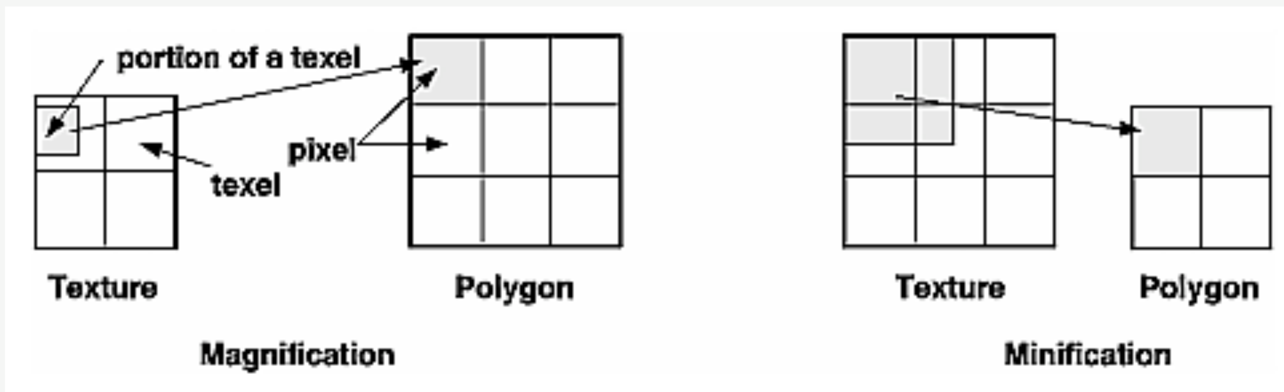
```
void GlTest::initTextures(){
    QImage buf, qtex;
    buf.load("/Users/dominikusbaur/texture.bmp");
    if(buf.isNull()){
        qDebug("could not load image");
        return;
    }
    qtex = QGLWidget::convertToGLFormat(buf);

    glGenTextures(1, &texture[0]);
    glBindTexture(GL_TEXTURE_2D, texture[0]);
    glTexImage2D(GL_TEXTURE_2D, 0, 3, qtex.width(), qtex.height(), 0,
                 GL_RGBA, GL_UNSIGNED_BYTE, qtex.bits());
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
}
```

```
glEnable(GL_TEXTURE_2D);

glBegin(GL_QUADS);
    glTexCoord2f(0, 0);
    glVertex3f(2, 2, 0);
    glTexCoord2f(1, 0);
    glVertex3f(-2, 2, 0);
    glTexCoord2f(1, 1);
    glVertex3f(-2, -2, 0);
    glTexCoord2f(0, 1);
    glVertex3f(2, -2, 0);
glEnd();
```



Texturen in OpenGL

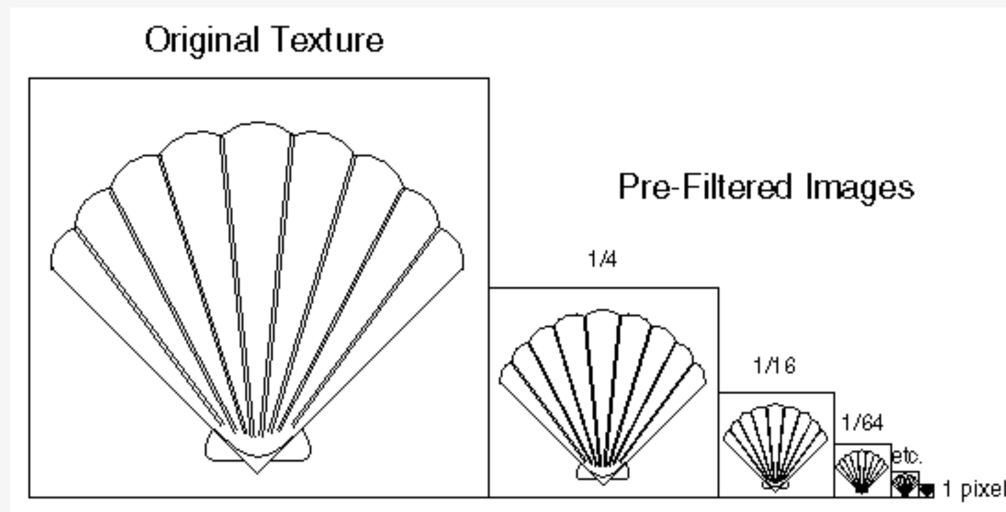
- Nachdem eine Textur auf ein Bildschirmobjekt gemappt wurde, kann ein Bildschirmpixel mehreren oder weniger als einem Texel entsprechen
- => Filterung (Minification bzw. Magnification) muss durchgeführt werden
- `glTexParameteri(target, pname, param)` stellt den Filter ein
  - `target` GL_TEXTURE_2D
  - `pname` GL_TEXTURE_MIN_FILTER, GL_TEXTURE_MAG_FILTER für Minification und Magnification
  - `param` Wert (GL_NEAREST, GL_LINEAR)



(Quelle: http://glprogramming.com/red/chapter09.html)

Mipmaps

- Texturierte Objekte die weiter von der Kamera entfernt sind brauchen nicht unbedingt die volle Texturauflösung. Um die Performance zu erhöhen können kleinere Varianten der Originaltextur dafür angegeben werden (mipmaps)
- `gluBuild2DMipmaps (target, internalformat, width, height, format, type, *pixels)`
  kann als Alternative zu glTexImage2D benutzt werden und generiert automatisch mipmaps



(Quelle: http://glprogramming.com/red/chapter09.html)

Blending

Blending

- Um Farbtransparenz in die OpenGL Berechnungen einfliessen zu lassen muss Alpha Blending aktiviert sein
- Beim Alpha Blending (bzw. Compositing) wird die (transparente) Farbe eines Objekts mit den existierenden Farben an der jeweiligen Position kombiniert um einen Transparenzeffekt zu erreichen
- Beleuchtung sollte deaktiviert sein
- Bei mehreren durchsichtigen Objekten ist die Zeichenreihenfolge wichtig - Aktivierung des Tiefenbuffers führt dazu, dass Objekte hinter einem transparenten Objekt nicht mehr gezeichnet werden!

(Quelle: http://www.opengl.org)

Blendingfunktionen

- Blending kombiniert Farben im Framebuffer (R_s, G_s, B_s, A_s) mit der aktuellen Farbe (R_d, G_d, B_d, A_d)
- Die Werte liegen zwischen 0 (vollständig transparent) und (k_R, k_G, k_B, k_A) (normalerweise 1 - vollständig opak) und werden mit einem Skalierungsfaktor (s für Source, d für Destination) multipliziert

$$R_d = \min(k_R, R_s s_R + R_d d_R)$$
$$G_d = \min(k_G, G_s s_G + G_d d_G)$$
$$B_d = \min(k_B, B_s s_B + B_d d_B)$$
$$A_d = \min(k_A, A_s s_A + A_d d_A)$$

Parameter	(f_R, f_G, f_B, f_A)
GL_ZERO	$(0, 0, 0, 0)$
GL_ONE	$(1, 1, 1, 1)$
GL_SRC_COLOR	$(R_s / k_R, G_s / k_G, B_s / k_B, A_s / k_A)$
GL_ONE_MINUS_SRC_COLOR	$(1, 1, 1, 1) - (R_s / k_R, G_s / k_G, B_s / k_B, A_s / k_A)$
GL_DST_COLOR	$(R_d / k_R, G_d / k_G, B_d / k_B, A_d / k_A)$
GL_ONE_MINUS_DST_COLOR	$(1, 1, 1, 1) - (R_d / k_R, G_d / k_G, B_d / k_B, A_d / k_A)$
GL_SRC_ALPHA	$(A_s / k_A, A_s / k_A, A_s / k_A, A_s / k_A)$
GL_ONE_MINUS_SRC_ALPHA	$(1, 1, 1, 1) - (A_s / k_A, A_s / k_A, A_s / k_A, A_s / k_A)$
GL_DST_ALPHA	$(A_d / k_A, A_d / k_A, A_d / k_A, A_d / k_A)$
GL_ONE_MINUS_DST_ALPHA	$(1, 1, 1, 1) - (A_d / k_A, A_d / k_A, A_d / k_A, A_d / k_A)$
GL_SRC_ALPHA_SATURATE	$(i, i, i, 1)$
GL_CONSTANT_COLOR	(R_c, G_c, B_c, A_c)
GL_ONE_MINUS_CONSTANT_COLOR	$(1, 1, 1, 1) - (R_c, G_c, B_c, A_c)$
GL_CONSTANT_ALPHA	(A_c, A_c, A_c, A_c)
GL_ONE_MINUS_CONSTANT_ALPHA	$(1, 1, 1, 1) - (A_c, A_c, A_c, A_c)$

(Quelle: http://pyopengl.sourceforge.net/documentation/manual/glBlendFunc.3G.html)

Blending in OpenGL

- `glEnable(GL_BLEND)` aktiviert das Blending
- `glBlendFunc(src, dest)` bestimmt die Blendingfunktion die benutzt wird
  - `src, dest` Ein Wert aus `GL_ZERO, GL_ONE, GL_DST_COLOR, GL_ONE_MINUS_DST_COLOR, GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA, GL_DST_ALPHA, GL_ONE_MINUS_DST_ALPHA, GL_SRC_ALPHA_SATURATE`
- `glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)` ist eine gängige Einstellung. Für $A_S = 1$ (keine Transparenz) ergeben sich dann folgende Ergebnisfarben:

$$R_d = R_s$$

$$G_d = G_s$$

$$B_d = B_s$$

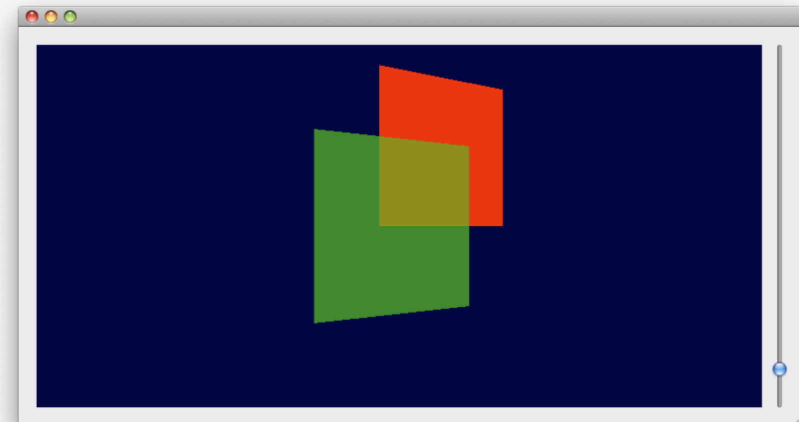
$$A_d = A_s$$

(Quelle: http://pyopengl.sourceforge.net/documentation/manual/glBlendFunc.3G.html)

Blending in OpenGL

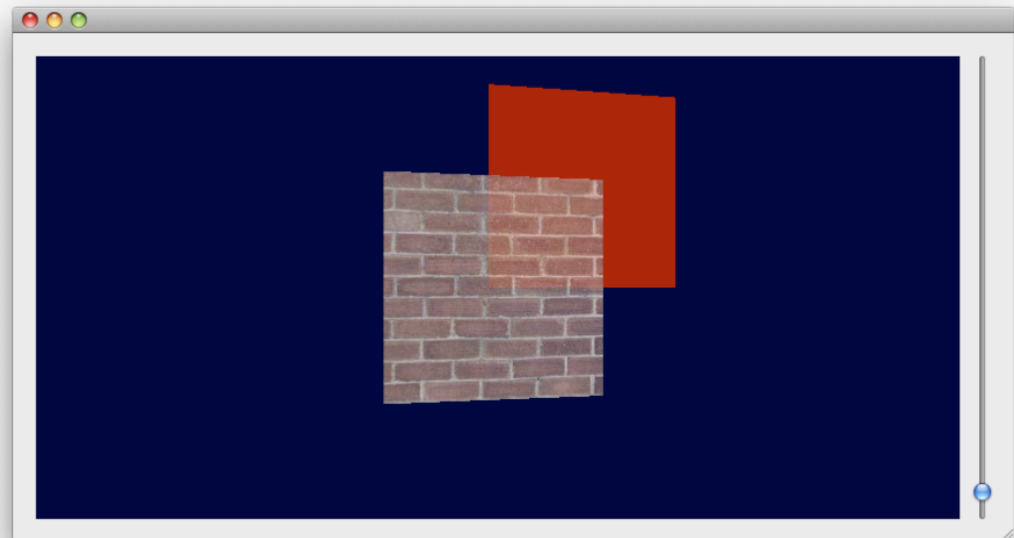
gltest.cpp

```
glDisable(GL_LIGHTING);
glEnable(GL_DEPTH_TEST);
glEnable(GL_BLEND);
glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

glColor4f(1.0f, 0.0f, 0.0f, 1.0f);
glBegin(GL_QUADS);
    glVertex3f(4, 4, -1);
    glVertex3f(0, 4, -1);
    glVertex3f(0, 0, -1);
    glVertex3f(4, 0, -1);
glEnd();

glColor4f(0.0f, 1.0f, 0.0f, 0.5f);
glBegin(GL_QUADS);
    glTexCoord2f(0, 0);
    glVertex3f(2, 2, 0);
    glTexCoord2f(1, 0);
    glVertex3f(-2, 2, 0);
    glTexCoord2f(1, 1);
    glVertex3f(-2, -2, 0);
    glTexCoord2f(0, 1);
    glVertex3f(2, -2, 0);
glEnd();
```



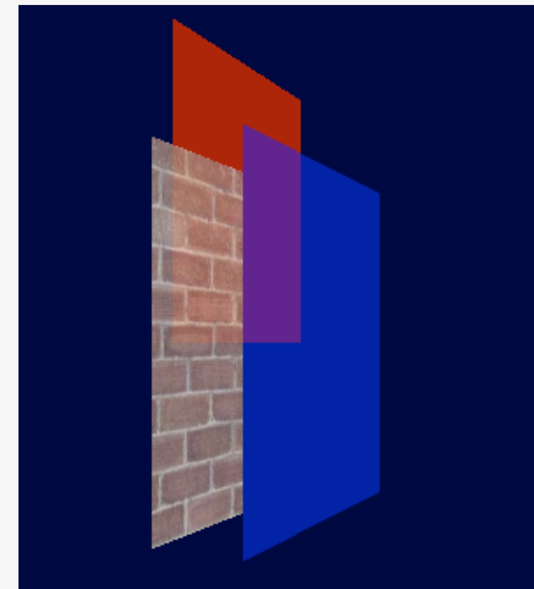
gltest.cpp

```
glColor4f(1.0f, 0.0f, 0.0f, 1.0f);
glBegin(GL_QUADS);
    glVertex3f(4, 4, -1);
    glVertex3f(0, 4, -1);
    glVertex3f(0, 0, -1);
    glVertex3f(4, 0, -1);
glEnd();

glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, texture[0]);
glColor4f(1.0f, 1.0f, 1.0f, 0.8f);
glBegin(GL_QUADS);
    glTexCoord2f(0, 0);
    glVertex3f(2, 2, 0);
    glTexCoord2f(1, 0);
    glVertex3f(-2, 2, 0);
    glTexCoord2f(1, 1);
    glVertex3f(-2, -2, 0);
    glTexCoord2f(0, 1);
    glVertex3f(2, -2, 0);
glEnd();
```



Blending – Best Practices

- Blending kann genauso wie Licht und Texturen zu- und abgeschaltet werden
- Einträge im Tiefenbuffer können verhindern dass Objekte hinter transparenten Objekten gezeichnet werden
- Daher:
  1. Tiefenbuffer aktivieren
  2. Alle opaken Objekte zeichnen
  3. Schreibschutz für Tiefenbuffer aktivieren:
     ```
     glDepthMask(GL_FALSE);
     ```
  4. Alle transparenten Objekte zeichnen (nach Tiefe sortiert)
  5. Schreibschutz wieder deaktivieren:
     ```
     glDepthMask(GL_TRUE);
     ```



(Quelle: http://users.informatik.uni-halle.de/~bugdoll/lehre/ws2007/opengl_hinweise_01.pdf)

Weiterführende Literatur

- http://www.opengl.org/sdk/docs/man/
- James Van Verth, Lars Bishop: Essential Mathematics for Games and Interactive Applications: A Programmer's Guide
- OpenGL ‘Redbook’: http://fly.srk.fer.hr/~unreal/theredbook/
- NeHe OpenGL Tutorials: http://nehe.gamedev.net/