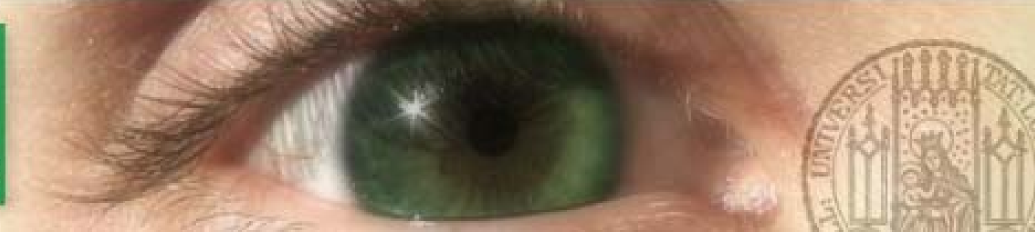
LMU

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

LEHR- UND FORSCHUNGSEINHEIT MEDIENINFORMATIK



Prof. Dr. Andreas Butz

Dipl.-Medieninf. Raphael Wimmer
Dipl.-Medieninf. Hendrik Richter

⑨

Übung: Computergrafik 1

Interaktion: Picking, Hit Testing

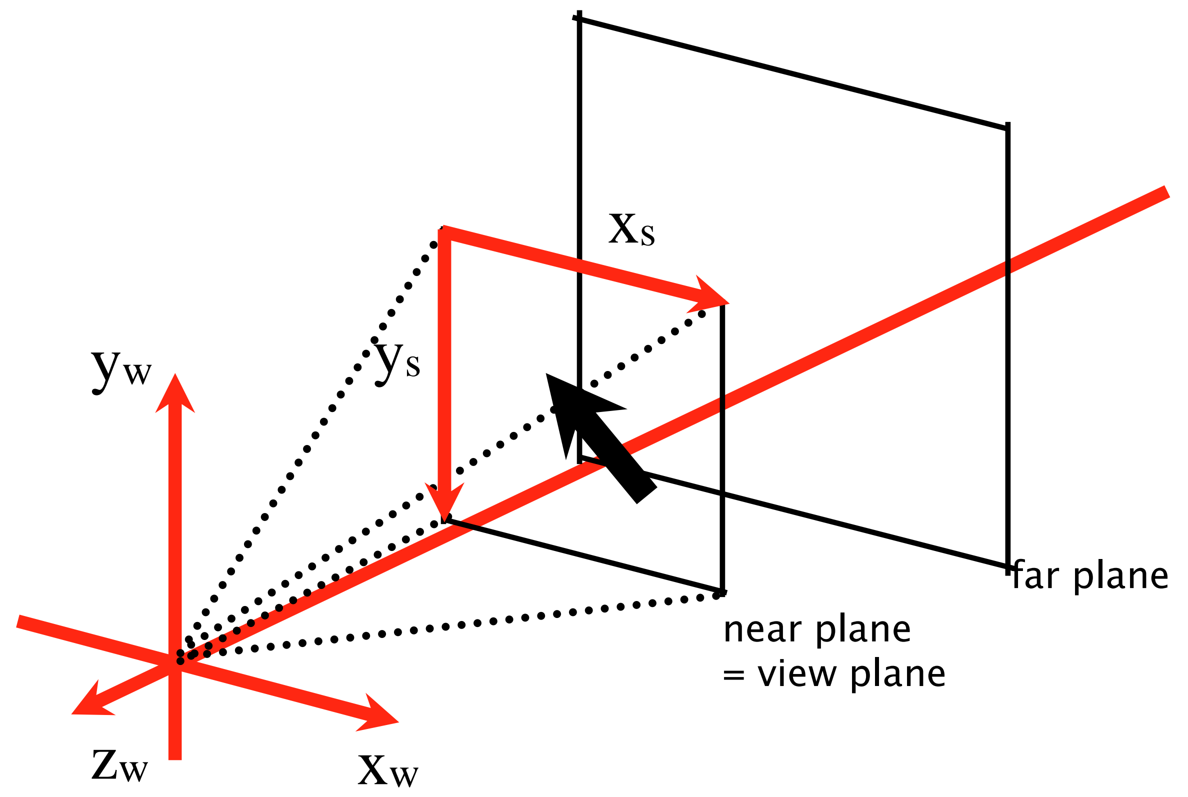
http://iparla.labri.fr/publications/2005/HK05/
HandheldLargeDisplay.JPG

Picking

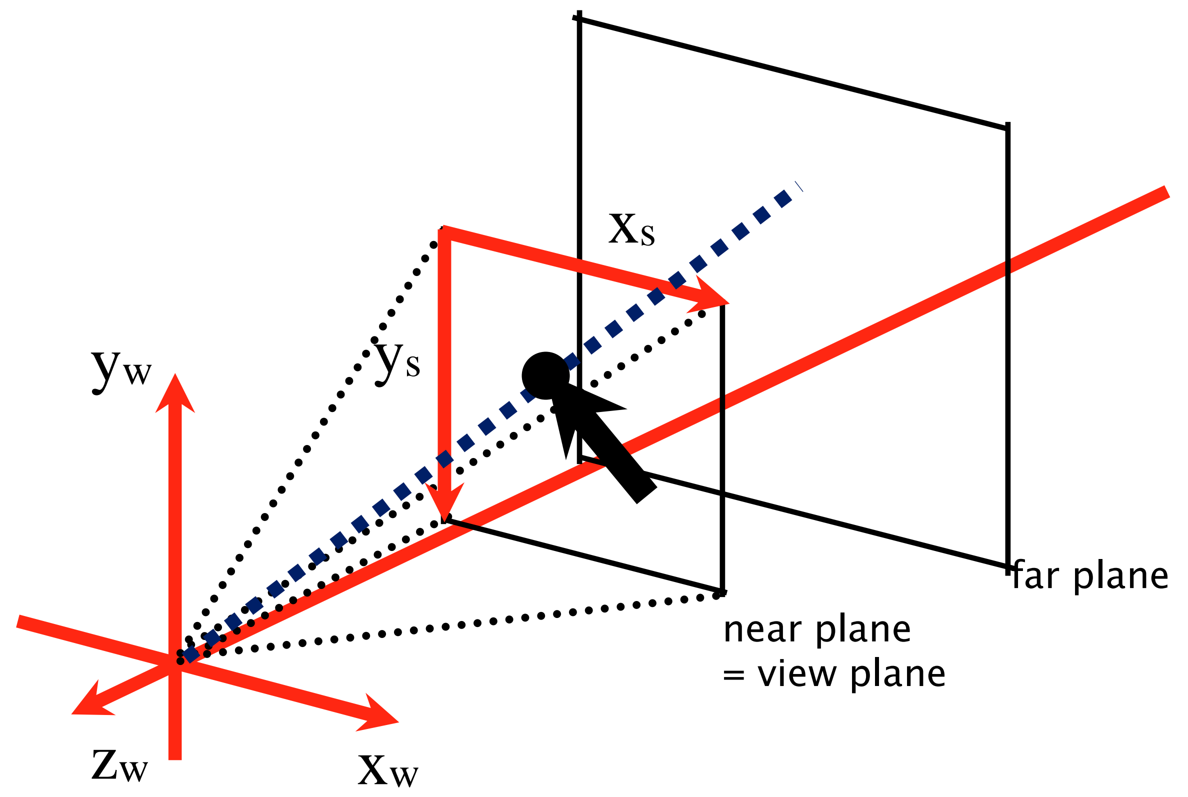
Picking

- Interaktion mit einer 3D-Szene erfolgt oft mit der Maus
- Dabei sind mehrere Aspekte zu beachten:
    - Mauskoordinaten sind im 2D-Bildschirmkoordinatensystem
    - Die Szene wird in 3D-Weltkoordinaten beschrieben
    - Die Tiefe der Objekte muss berücksichtigt werden - der Benutzer will vermutlich das Objekt auswählen, das am nächsten an der Kamera liegt
    - Die Form der Objekte spielt ebenfalls eine Rolle

Picking - Schematisch



Picking - Schematisch



OpenGL Transformationen - Picking

- Erster Schritt: Konvertierung der 2D-Mauskoordinaten in 3D-Weltkoordinaten
- Vor der Darstellung der Szene wurde genau dasselbe in der umgekehrten Reihenfolge durchgeführt

Rendering

	Model-View Matrix		Projection Matrix		Perspective Division		Viewport Transformation	
Object coordinates	→	Eye coordinates	→	Clip coordinates	→	Normalized Device coordinates	→	Window coordinates
	GL_MODELVIEW		GL_PROJECTION				glViewport	
Object coordinates	←	Eye coordinates	←	Clip coordinates	←	Normalized Device coordinates	←	Window coordinates (x, y)

Picking

(Quelle: http://www.opengl.org/resources/faq/technical/transformations.htm)

glu(Un)Project

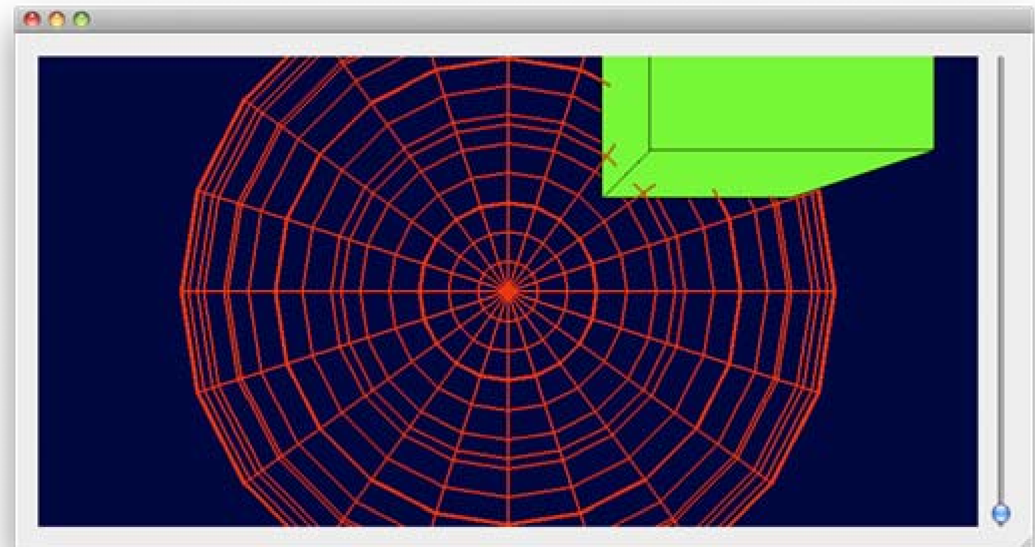
- GLU enthält die Funktionen `gluProject` und `gluUnProject` um zwischen Fenster- und Weltkoordinaten zu konvertieren
- `glu(Un)Project (inpX, inpY, inpZ, model, proj, view, outX, outY, outZ)`
  - `inpX, inpY, inpZ` Punkt im Ausgangskoordinatensystem
  - `model` Modelview-Matrix (sechzehnstelliges Array)
  - `proj` Projection-Matrix (sechzehnstelliges Array)
  - `view` Viewport-Vektor (vierstelliges Array)
  - `outX, outY, outZ` Konvertierter Punkt im Zielkoordinatensystem
- `gluProject` konvertiert von Welt- zu Fensterkoordinaten (normale OpenGL-Transformationspipeline)
- `gluUnProject` konvertiert von Fenster- zu Weltkoordinaten (invertierte OpenGL-Matrizen)

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

Picking mit gluUnProject

gltest.cpp

```
GLfloat spherepos[];

void GlTest::paintGL(){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0, 0, -10);
    glRotatef(rotaY, 0, 1, 0);

    glEnable(GL_DEPTH_TEST);
    glPushMatrix();
    glColor3f(0, 1, 0);
    glTranslatef(2, 2, 5);
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    initABox();
    glColor3f(0,0,0);
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    glLineWidth(2);
    initABox();
    glPopMatrix();

    glPushMatrix();
    glTranslatef(spherepos[0], spherepos[1], spherepos[2]);
    glColor3f(1, 0, 0);
    GLUquadricObj* quad = gluNewQuadric();
    gluSphere(quad, 5, 20, 20);
    glPopMatrix();
}
```



Picking mit gluUnProject

gltest.h

```
protected:
    // overrides:
    void initializeGL();
    void paintGL();
    void resizeGL(int width, int height);

    void mousePressEvent(QMouseEvent *);
    void mouseMoveEvent(QMouseEvent *);

    void setSphere(QPoint mousepos);

private:
    GLdouble* picking(QPoint mousepos);
```

gltest.cpp

```
void GlTest::setSphere(QPoint mousepos){
    GLdouble* target = picking(mousepos);

    spherepos[0] = target[0];
    spherepos[1] = target[1];
    spherepos[2] = target[2];

    updateGL();
}

void GlTest::mousePressEvent(QMouseEvent *me){
    setSphere(me->pos());
}

void GlTest::mouseMoveEvent(QMouseEvent *me){
    setSphere(me->pos());
}
```

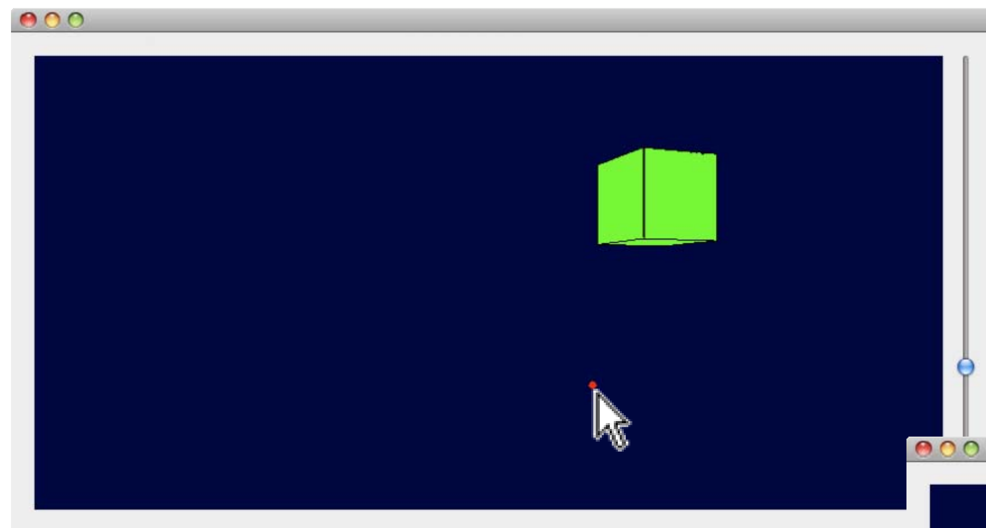
Picking mit gluUnProject

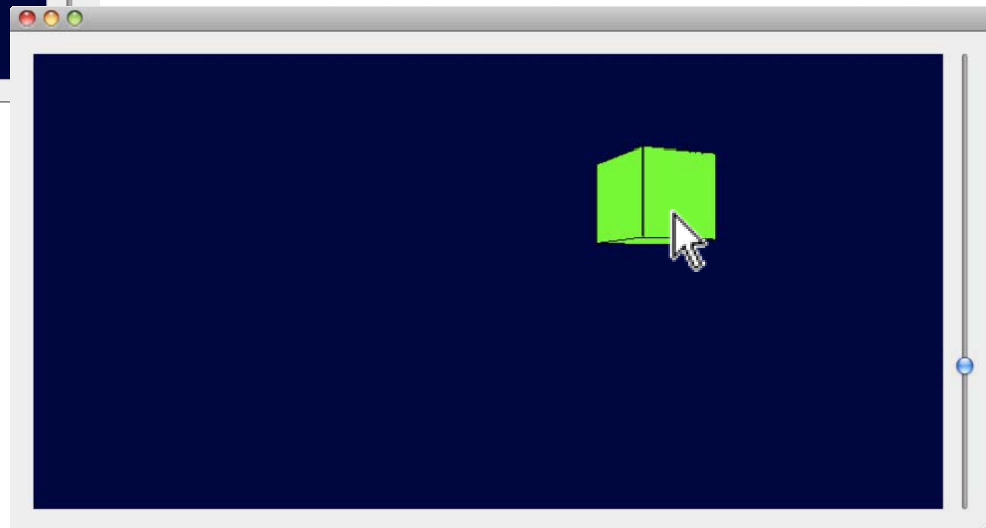
gltest.cpp

```
GLdouble* GlTest::picking(QPoint mousepos){

    GLint viewport[4];
    GLdouble modelview[16];
    GLdouble projection[16];
    GLfloat winX, winY, winZ;
    GLdouble pos[] = {0,0,0};

    glGetDoublev(GL_MODELVIEW_MATRIX, modelview);
    glGetDoublev(GL_PROJECTION_MATRIX, projection);
    glGetIntegerv(GL_VIEWPORT, viewport);

    winX = (GLfloat) mousepos.x();
    winY = viewport[3] - mousepos.y();
    winZ = 1.0;
    gluUnProject(winX, winY, winZ, modelview, projection, viewport,
                 &pos[0], &pos[1], &pos[2]);

    return pos;
}
```

(Quelle: http://nehe.gamedev.net/data/articles/article.asp?article=13)

Picking mit gluUnProject





glReadPixels

- winZ = 1.0 führt bei gluUnProject dazu, dass die Position des Mauspfeils auf die far clipping plane gemappt wird - Verdeckung!
- Tiefeninformationen sind im Depth Buffer gespeichert!
- glReadPixels erlaubt das Auslesen des OpenGL Frame Buffer:
- glReadPixels (x, y, width, height, format, type, *pixels)
  - x,y Koordinaten des Punkts im Buffer
  - width, height Größe des auszulesenden Bereichs (width=height=1 für 1 Pixel)
  - format Format der Pixeldaten (z.B. GL_DEPTH_COMPONENT, GL_RGB)
  - type Datentyp der Pixeldaten (z.B. GL_FLOAT, GL_UNSIGNED_BYTE)
  - *pixels Zielpointer für die Pixeldaten

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

glReadPixels + gluUnProject

gltest.cpp

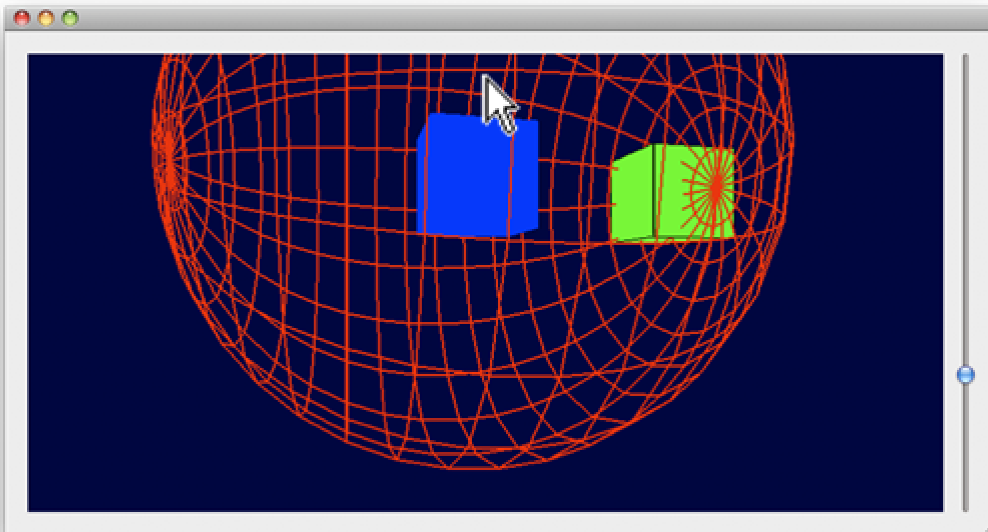
```
GLdouble* GlTest::picking(QPoint mousepos){

    GLint viewport[4];
    GLdouble modelview[16];
    GLdouble projection[16];
    GLfloat winX, winY, winZ;
    GLdouble pos[] = {0,0,0};

    glGetDoublev(GL_MODELVIEW_MATRIX, modelview);
    glGetDoublev(GL_PROJECTION_MATRIX, projection);
    glGetIntegerv(GL_VIEWPORT, viewport);

    winX = (GLfloat) mousepos.x();
    winY = viewport[3] - mousepos.y();

    glReadPixels(winX, winY, 1, 1, GL_DEPTH_COMPONENT, GL_FLOAT, &winZ);
    gluUnProject(winX, winY, winZ, modelview, projection, viewport,
                 &pos[0], &pos[1], &pos[2]);

    return pos;
}
```

glReadPixels + gluUnProject

glReadPixels für Farbe

gltest.cpp

```
void GlTest::mousePressEvent(QMouseEvent *me){
    GLint viewport[4];
    GLubyte col[4];
    glGetIntegerv(GL_VIEWPORT, viewport);
    glReadPixels(me->pos().x(), viewport[3] - me->pos().y(), 1, 1, GL_RGBA, GL_UNSIGNED_BYTE, &col);
    qDebug("r:%d g:%d b:%d a:%d", col[0], col[1], col[2], col[3]);

    setSphere(me->pos());
}
```

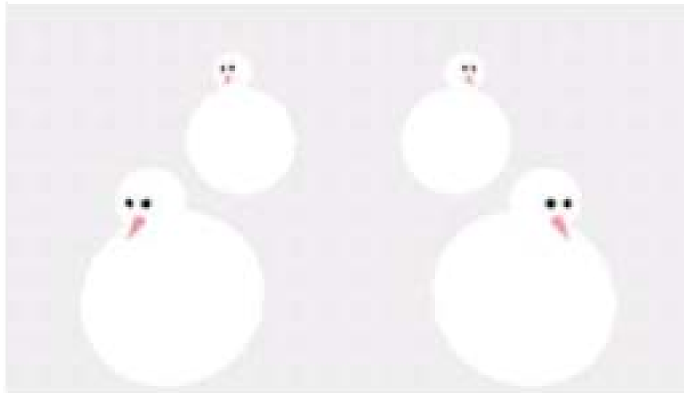


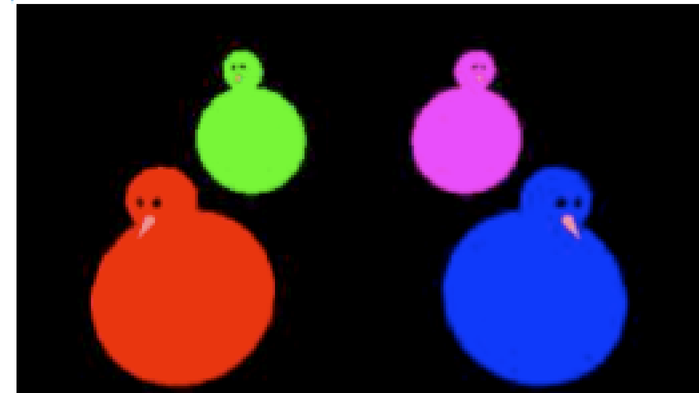
Output:

```
r:0 g:0 b:51 a:255
r:0 g:0 b:255 a:255
r:0 g:255 b:0 a:255
r:255 g:0 b:0 a:255
r:0 g:0 b:255 a:255
```

Backbuffer Color Coding

- Getroffene Objekte lassen sich also anhand ihrer Farbe unterscheiden. Problem: Shading, realistische Farben, etc.
- Daher: Rendering in den Backbuffer (dazu muss Double Buffering aktiv sein)
- Für Details s. Quelle

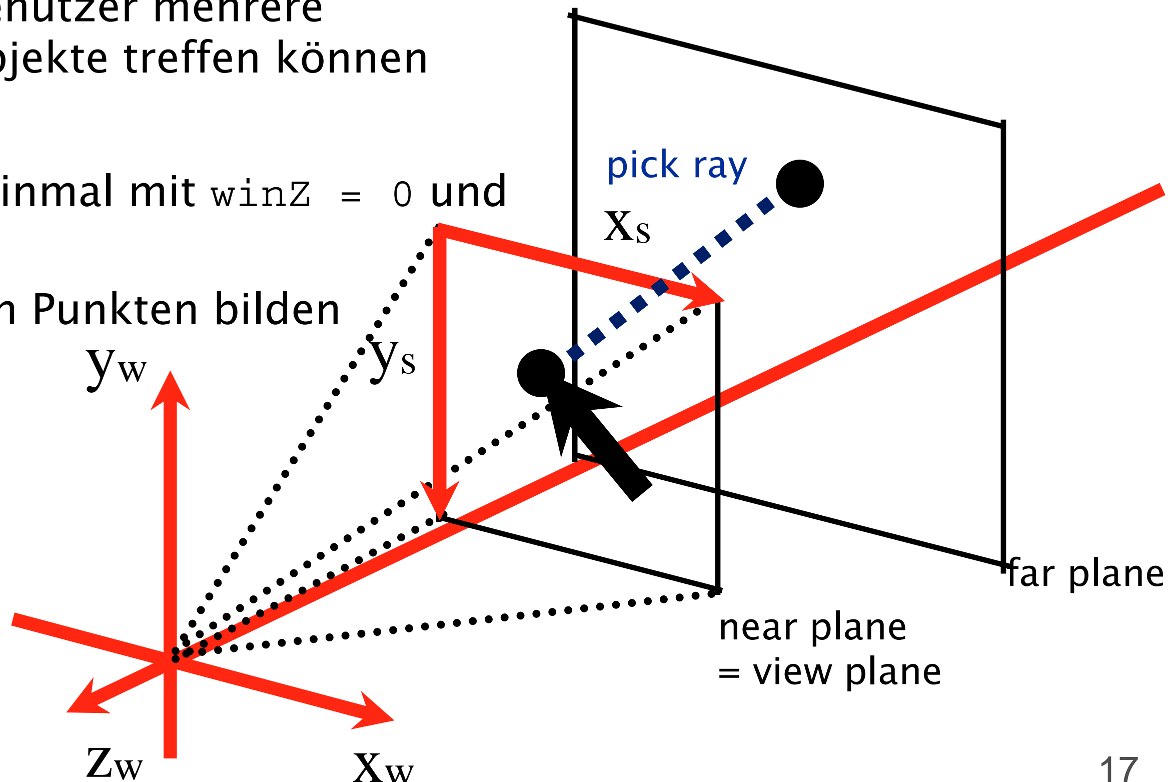




(Quelle: http://www.ugrad.cs.ubc.ca/~cs314/Vjan2007/schedule.html#week10)

Pick Rays

- Das Auslesen des Depth Buffers führt nicht unbedingt zum gewünschten Ergebnis:
  - Transparente Pixel werden normalerweise ebenfalls in den Depth Buffer geschrieben
  - Möglicherweise soll der Benutzer mehrere hintereinanderliegende Objekte treffen können
- Lösung: Pick Rays.
  - Zweimal `gluUnProject`, einmal mit `winZ = 0` und einmal mit `WinZ = 1`
  - Dann Gerade zwischen den Punkten bilden



Pick Rays mit gluUnProject

gltest.cpp

```
GLdouble* GlTest::pickRay(QPoint mousepos){

    GLint viewport[4];
    GLdouble modelview[16];
    GLdouble projection[16];
    GLfloat winX, winY, winZ;
    GLdouble pickray[] = {0,0,0, 0,0,0};

    glGetDoublev(GL_MODELVIEW_MATRIX, modelview);
    glGetDoublev(GL_PROJECTION_MATRIX, projection);
    glGetIntegerv(GL_VIEWPORT, viewport);

    winX = (GLfloat) mousepos.x();
    winY = viewport[3] - mousepos.y();
    winZ = 0;
    gluUnProject(winX, winY, winZ, modelview, projection, viewport,
                 &pickray[0], &pickray[1], &pickray[2]);
    winZ = 1;
    gluUnProject(winX, winY, winZ, modelview, projection, viewport,
                 &pickray[3], &pickray[4], &pickray[5]);

    return pickray;

}
```

Pick Rays mit gluUnProject

Bounding Volumes

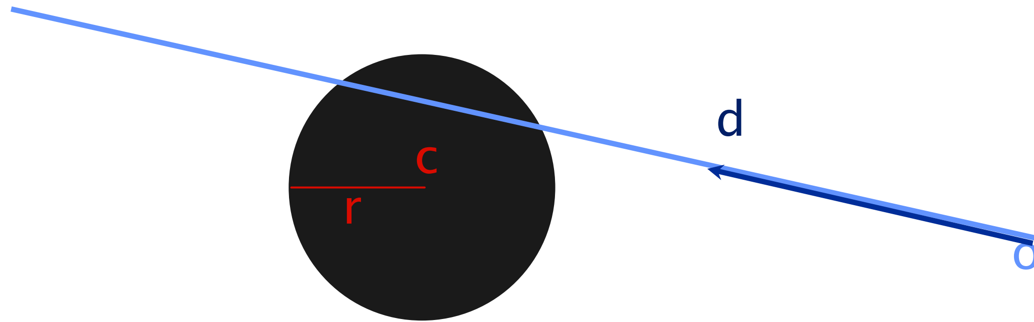
Hit Testing

- Eine exakte Bestimmung ob ein virtueller Körper getroffen wurde ist meist zu aufwändig - dabei müsste eine Berechnung für jedes gezeichnete Polygon durchgeführt werden
- Daher behilft man sich mit sogenannten Bounding Volumes
- Ein Bounding Volume ist ein einfacher geometrischer Körper der das zu testende Objekt möglichst eng umschließt
- Oft verwendete Bounding Volumes sind Bounding Sphere und Bounding Box (auch Axis-Aligned)

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

Hit Testing - Ray+Sphere

- Einfaches Hittesting lässt sich mit einer Bounding Sphere durchführen
- Falls der Strahl die Kugel schneidet liegt ein Treffer vor, sonst nicht
- Berechnung lässt sich algebraisch oder geometrisch durchführen

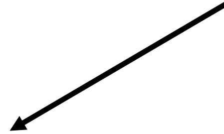


(Quelle: http://devmaster.net/wiki/Ray-sphere_intersection)

Hit Testing - Ray+Sphere

- Algebraisch: Kugel und Strahl werden als Gleichungen dargestellt

$$(\mathbf{x}-\mathbf{c})\cdot(\mathbf{x}-\mathbf{c}) = r^2 \qquad \mathbf{x}(t) = \mathbf{o} + t\mathbf{d}$$



$$(\mathbf{o}+t\mathbf{d}-\mathbf{c})\cdot(\mathbf{o}+t\mathbf{d}-\mathbf{c}) = r^2$$

- Nach t auflösen => quadratische Gleichung!

$$At^2 + Bt + C = 0 \qquad \begin{aligned} A &= \mathbf{d}\cdot\mathbf{d}, \\ B &= 2(\mathbf{o}-\mathbf{c})\cdot\mathbf{d}, \\ C &= (\mathbf{o}-\mathbf{c})\cdot(\mathbf{o}-\mathbf{c}) - r^2 \end{aligned}$$

- Lösung durch “Mitternachtsformel”. t > 0 bedeutet, dass es einen gültigen Schnittpunkt gibt. |d| = 1, damit fällt A weg.

(Quelle: http://devmaster.net/wiki/Ray-sphere_intersection)

Hit Testing - Ray+Sphere

- In Code:

```
float intersectRaySphere(const Ray &ray, const Sphere &sphere) {
    float* dst = sub(ray.o, sphere.c);
    float B = dot(dst, ray.d);
    float C = dot(dst, dst) - (sphere.r * sphere.r);
    float D = B*B - C;
    return D > 0 ? -B - sqrt(D) : std::numeric_limits<float>::infinity();
}
```

- `dot(v1, v2)` ist das Skalarprodukt

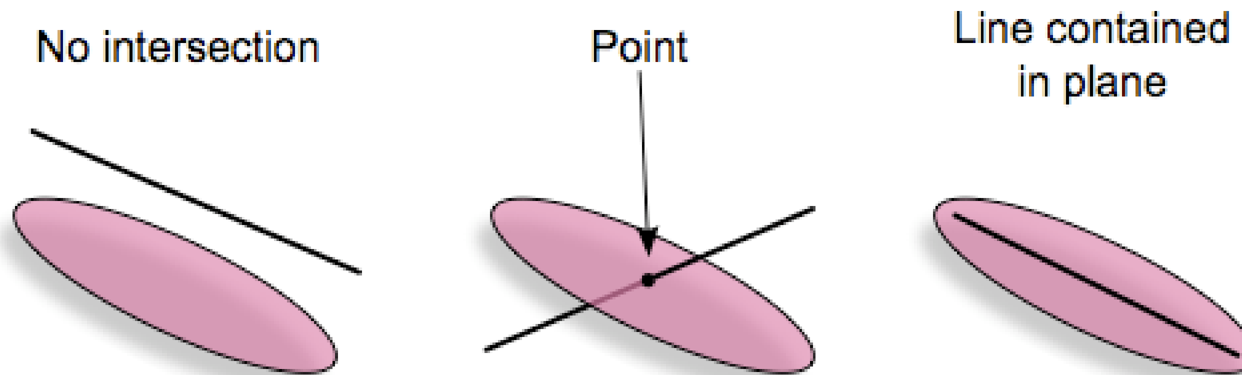
$$A = \mathbf{d} \cdot \mathbf{d},$$
$$B = 2(\mathbf{o} - \mathbf{c}) \cdot \mathbf{d},$$
$$C = (\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - r^2$$

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

(Quelle: http://devmaster.net/wiki/Ray-sphere_intersection)

Hit Testing - Ray+Plane

- Ein weiteres wiederkehrendes Problem ist den Schnittpunkt des Strahls mit einer Ebene zu finden
- Drei verschiedene Fälle: Kein Schnittpunkt, ein Schnittpunkt oder unendlich viele (Gerade liegt in Ebene)
- Berechnung lässt sich wieder algebraisch oder geometrisch durchführen



(Quelle: http://en.wikipedia.org/wiki/Line-plane_intersection)

Hit Testing - Ray+Plane

- Algebraisch: Ebene und Strahl werden als Gleichungen dargestellt

$$\mathbf{p} \cdot \mathbf{n} = d \qquad \mathbf{l}_a + (\mathbf{l}_b - \mathbf{l}_a)t, \quad t \in \mathbb{R},$$

$$(\mathbf{l}_a + t(\mathbf{l}_b - \mathbf{l}_a)) \cdot \mathbf{n} = d,$$

$$\mathbf{n} = (a, b, c) \qquad t = \frac{d - \mathbf{l}_a \cdot \mathbf{n}}{(\mathbf{l}_b - \mathbf{l}_a) \cdot \mathbf{n}}.$$

$$t = \frac{d - ax_a - by_a - cz_a}{a(x_b - x_a) + b(y_b - y_a) + c(z_b - z_a)}.$$

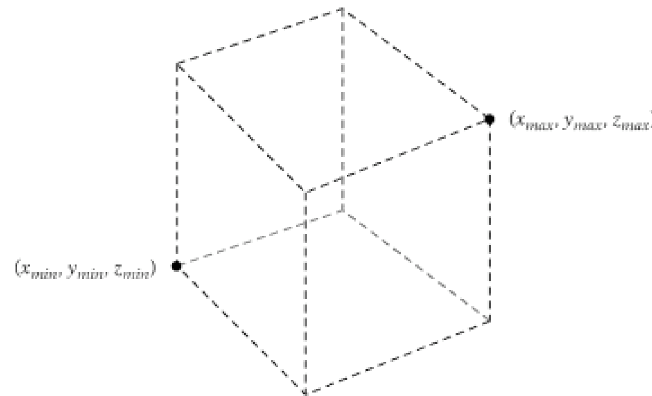
Lösungen:

- Schnittpunkt
- Nenner = 0: Gerade ist parallel zur Ebene
- Zähler = Nenner = 0: Gerade liegt in der Ebene

(Quelle: http://en.wikipedia.org/wiki/Line-plane_intersection)

Hit Testing - Bounding Boxes

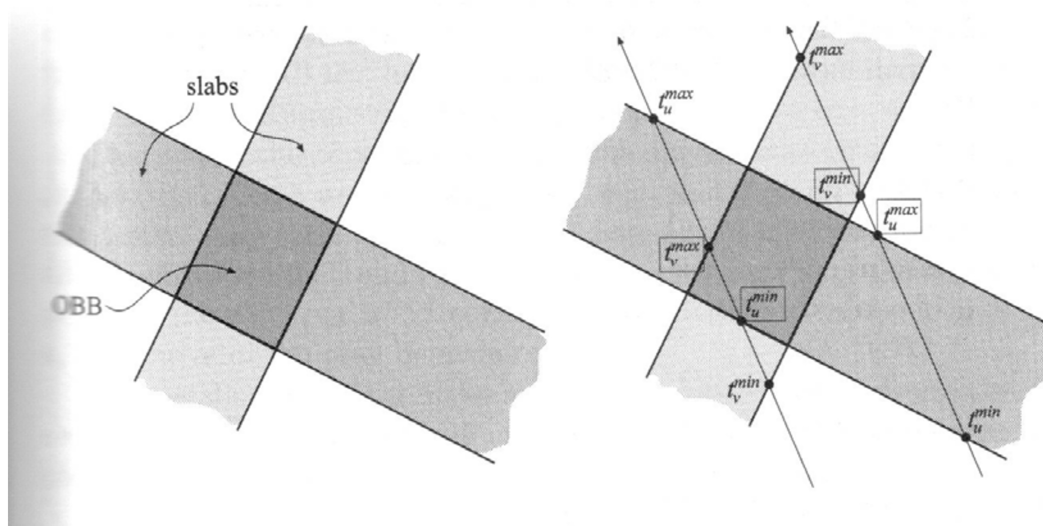
- Für manche Körper ist eine Bounding Box eher geeignet
- Dabei wird zwischen willkürlich liegenden Oriented Bounding Boxes (OBBs) und an den Koordinatenachsen ausgerichteten Axis-Aligned Bounding Boxes (AABBs) unterschieden
- Um die Berechnungen zu vereinfachen bietet sich eine AABB an
- Diese wird durch ihren minimalen Punkt B1 und ihren maximalen Punkt B2 definiert



(Quelle: Verth: Essential Mathematics for Games and Interactive Applications)

Hit Testing - Ray+AABB

- “Slabs” sind die Flächen zwischen zwei Ebenen
- Eine dreidimensionale Bounding Box beschreibt insgesamt drei Slabs, eine für jede Dimension
- Idee: Der Strahl schneidet die Bounding Box wenn er Punkte enthält die ebenfalls in *allen* drei Slabs enthalten sind

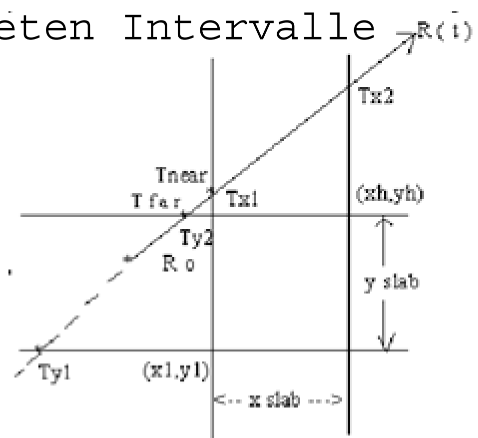


(Quelle: http://modsim.gist.ac.kr/lecture/2009_spring/April29.ppt)

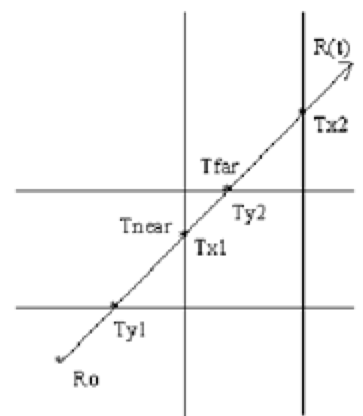
Hit Testing - Ray+AABB

- Idee für den Test-Algorithmus:

- Für jeden Slab:
  - Löse die Geradengleichung für den minimalen und maximalen Wert des Slabs (also die begrenzenden Ebenen)
    => Intervall innerhalb der Gerade
- Überprüfe ob sich die drei berechneten Intervalle überschneiden
  => Falls ja, Strahl schneidet AABB
  => Falls nein, kein Schnittpunkt



Tnear > Tfar, so ray misses box

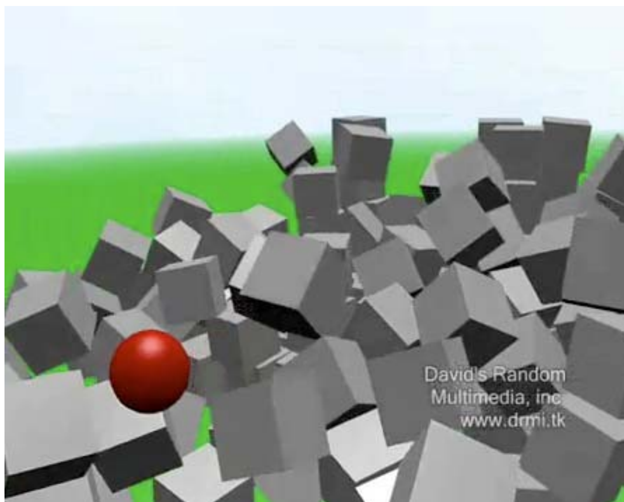


Tnear < Tfar, & Tnear > 0, so Tnear is intersection distance

(Quelle: http://www.siggraph.org/education/materials/HyperGraph/raytrace/rtinter3.htm)

Hit Testing - Weitere Anwendungen

- Weitere wichtige Anwendung von Hit Tests: Kollisionserkennung!
- Berechnung von weiteren Schnittpunkten für andere geometrische Körper (Kugel<->Ebene, Kugel<->Zylinder, Kugel<->Box, etc).
- Rechnerisch aufwändig, pro Frame Vergleich jedes mit jedem anderen Objekt
- Inzwischen auch Hardware-beschleunigt (nVidia PhysX)

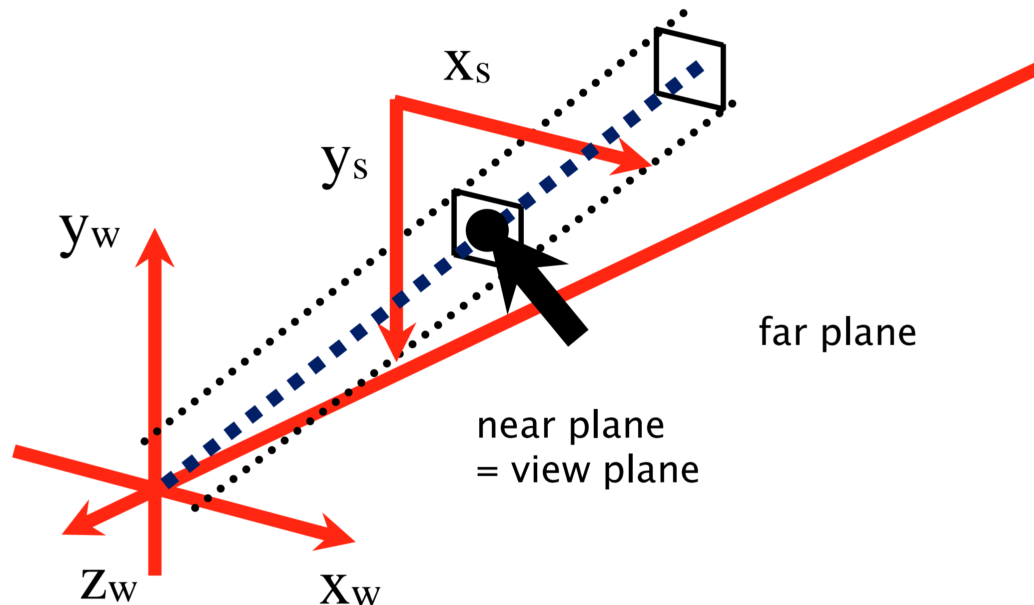


*(Quellen: http://www.medien.ifi.lmu.de/lehre/ss08/3dp/
http://www.youtube.com/watch?v=GK2Bukrmr4s
http://www.nvidia.com/object/nvidia_physx.html)*

Picking mit GL_SELECT

Picking - GL_SELECT

- Grundidee: Verkleinerung des Frustrums auf die Größe des Pick Rays => alle gezeichneten Objekte wurden getroffen
- OpenGL bietet Befehle an um Objekte zu benennen und getroffene ausgeben zu lassen



GL_SELECT

- `glRenderMode` bestimmt den aktuellen Zeichenmodus. `GL_RENDER` ist voreingestellt, wir brauchen `GL_SELECT`
- Im `GL_SELECT` Modus wird nichts gezeichnet, aber alle Objekte die gezeichnet würden werden in einen mit `glSelectBuffer` vorgegebenen Buffer geschrieben:
- `glSelectBuffer(int size, *buffer)`

- Der name stack enthält Namen für getroffene Objekte:
- `glInitNames()` initialisiert den name stack
- `glPushName(int name)` legt einen Namen auf den name stack
- `glLoadName(int name)` setzt für alle folgenden Objekte einen Namen

- Ein Aufruf von `glRenderMode(GL_RENDER)` gibt die Anzahl der getroffenen Objekte zurück

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

GL_SELECT

```
GLuint buffer[512];
GLint hits;

void GlTest::selection(float mouse_x, float mouse_y){
    GLint       viewport[4];
    glGetIntegerv(GL_VIEWPORT, viewport);

    glSelectBuffer(512, buffer);

    glRenderMode(GL_SELECT);
    glInitNames();
    glPushName(0);

    glMatrixMode(GL_PROJECTION);
    glPushMatrix();
    glLoadIdentity();
```

(Quelle: http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=32)

GL_SELECT + glu

- Um das View Frustum zu verkleinern nutzen wir Hilfsfunktionen aus GLU
- `gluPickMatrix(x, y, width, height, viewport)` erzeugt eine passende Projektionsmatrix
  - `x,y` Koordinaten in Fensterkoordinaten (Vorsicht: y-Achse spiegeln!)
  - `width, height` Größe des Frustums (meist: width = height = 1)
  - `viewport` Der aktuelle Viewport
- Setzen der richtigen Perspektive mit `gluPerspective`

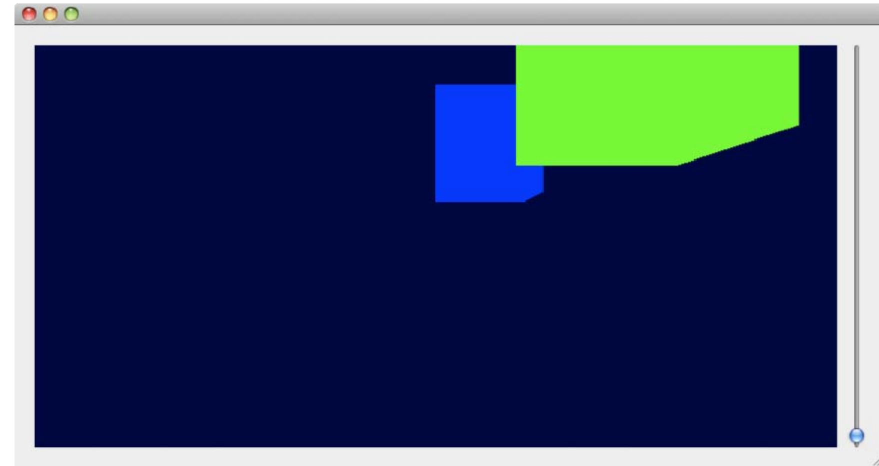
(Quelle: http://www.opengl.org)

GL_SELECT

```
gluPickMatrix((GLdouble) mouse_x, (GLdouble) (viewport[3]-mouse_y), 1.0f, 1.0f, viewport);

gluPerspective(45.0f, (GLfloat) (viewport[2]-viewport[0])/(GLfloat) (viewport[3]-viewport[1]), 0.1f, 100.0f);
glMatrixMode(GL_MODELVIEW);

drawScene();

glMatrixMode(GL_PROJECTION);
glPopMatrix();
glMatrixMode(GL_MODELVIEW);
hits=glRenderMode(GL_RENDER);
```

(Quelle: http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=32)

GL_SELECT



```
void GlTest::drawScene(){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0, 0, -10);
    glRotatef(rotaY, 0, 1, 0);

    glEnable(GL_DEPTH_TEST);
    glLoadName(1);
    glPushMatrix();
    glTranslatef(1, 2, 0);
    glColor3f(0, 0, 1);
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    initABox();
    glPopMatrix();

    glLoadName(2);
    glPushMatrix();
    glColor3f(0, 1, 0);
    glTranslatef(2, 2, 5);
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    initABox();
    glPopMatrix();
}
```

(Quelle: http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=32)

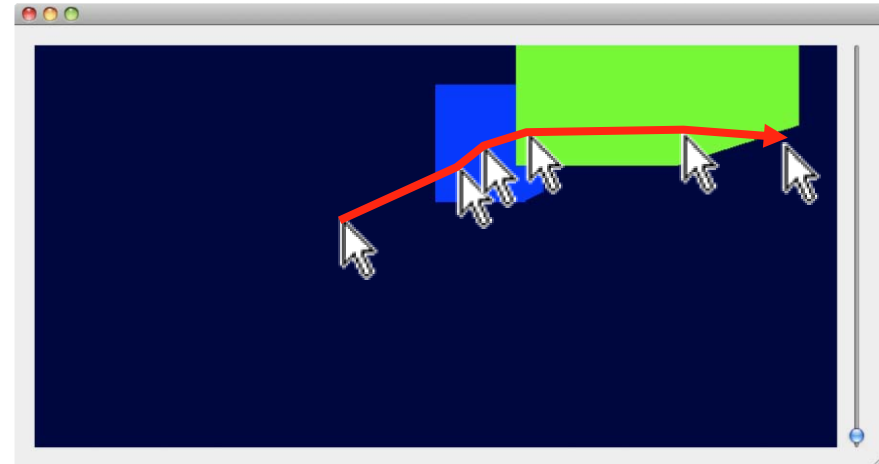
GL_SELECT Hit Buffer

- glRenderMode(GL_RENDER) gibt die Anzahl der getroffenen Elemente zurück
- Für jedes dieser Elemente werden vier Einträge im gesetzten Buffer gemacht:
  - 0 Anzahl der Elemente im name stack beim Treffer
  - 1 Minimaler z-Wert für den getroffenen Vertex
  - 2 Maximaler z-Wert für den getroffenen Vertex
  - 3 Der Name des Vertex im name stack

- Vergleich der z-Werte muss händisch gemacht werden

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

GL_SELECT



```
if(hits > 0){
    int minname = buffer[3];
    int mindist = buffer[1];
    for(int i = 1; i < hits; i++){
        if(buffer[i * 4 + 1] < mindist){
            mindist = buffer[i * 4 + 1];
            minname = buffer[i * 4 + 3];
        }
    }

    qDebug("hit object: %d", minname);
} else {
    qDebug("hit no object.");
}
```

gltest.cpp

Output:

hit no object.
hit object: 1
hit object: 1
hit object: 2
hit object: 2
hit no object.

(Quelle: http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=32)

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/