Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

Introduction to Computer Graphics

Prof. Dr. David Strippgen

Exercise 2

Stars & Coordinate Systems

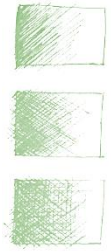
Lernziele

- Der/die Studierende kann
  - 3D statt 2D (Projektionsabbildung)
  - Camera Steuerung mit gluLookAt() benutzen
  - Weiterführende OpenGL Befehle glTranslate, glRotate, glScale benutzen
  - Depth (Z-Buffer) einschalten
  - Push/Pop Matrix (Hierarchische Modelle)
  - Lichter benutzen
- Die Übung besteht aus 4 Teilaufgaben

Eine interaktive Applikation

```
public class Ex2Stars {

    boolean endThisApp = false;

    public static void main(String[] args) {
        Ex2Stars app = new Ex2Stars();
        app.run();
    }

    public void run(){
        init();
        while(!endThisApp){
            update();
            draw();
        }
    }

    public void init(){
        //DS Everything that needs to be done once!
    }
    public void update(){
        //DS Update the state of your world, get user input, etc.
    };
    public void draw(){
        //DS Render your meshes
    };

}
```

OpenGL Reprise



- In OpenGL gibt es zwei wichtige Matrizen:
  - **ModelView und Projection**!
- Wir werden ModelView als die Matrix verstehen in der **ALLE** Transformationen bis auf die Projektion stattfinden.
- Also muss **lookAt()** auch in MODELVIEW stattfinden.
- Erklärung: Siehe auch hier!

Settings to be done once

- In init()
  - öffnen wir einen Display und
  - setzen wir die grundlegenden OpenGL Settings
  - setzen der Hintergrundfarbe
    - **GL11.glClearColor(0.0f, 0.0f, 0.5f, 0.0f);**

- **neu**:
- GL11.glMatrixMode(GL11.*GL_**PROJECTION**);*
  GL11.**glLoadIdentity**(); // **Saubermachen!**
  GLU.**gluPerspective**(45.f, w/(float)h, 0.1f, 3000.f);
  // Ist neu: Statt glOrtho()

Benötigt Field GLU aus **lwjgl_util.jar** im BuildPath

In draw() zeichnen wir unsere OpenGL Zeichnung:

- Als erstes: Sauber machen, den letzten Frame löschen mit:

  GL11.glClear(GL11.*GL_COLOR_BUFFER_BIT*);

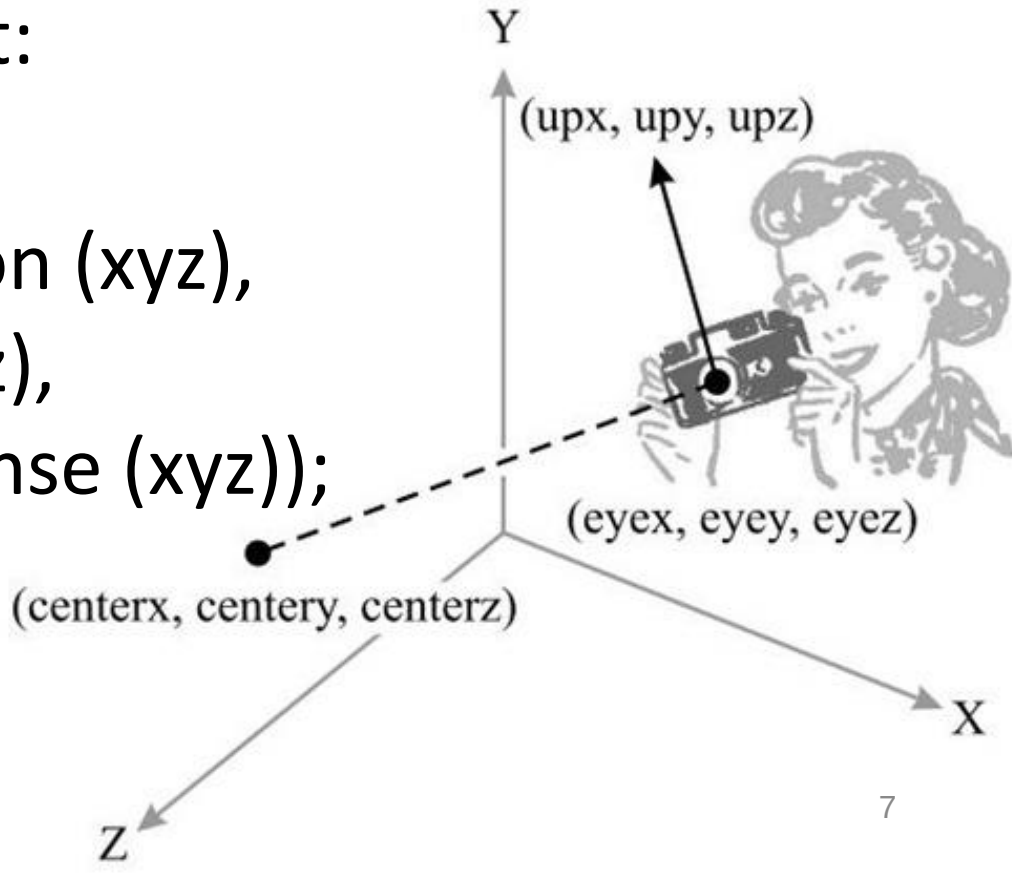
Und auch die Matrix will gelöscht werden:

- **GL11.glMatrixMode(GL11.*GL_MODELVIEW*);**

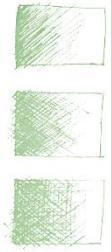
  GL11.glLoadIdentity();

Draw…

- Nachdem wir sauber gemacht haben, müssen wir erst mal sagen, wo die Kamera stehen soll und wo sie hinschaut:
- gluLookAt(
  Kamera Position (xyz),
  Blickpunkt (xyz),
  Nach Oben Achse (xyz));



Setting the Camera Position



- Dann wollen wir nun die Transformation für die CameraSpace-Abbildung laden:

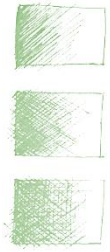
- **GLU.gluLookAt(cameraPos[0],cameraPos[1], cameraPos[2], 0,0,0, 0,1,0);**

- Camera steht hier an
- **float[] cameraPos = new float[]{0,0,10};**
- Camera schaut auf den Ursprung **(0,0,0)**
- Y-Achse ist UP **(0,1,0)**

Übung 2 – 3D Welt

- Wir legen eine neue Klasse Ex2Stars an, wie angeben
- Füllen Sie die **init**():
  - Display init und gluPerspective params setzen
- In **draw**()
  - Clear Screen / Clear Matrix und Display.update()
- In **update**()
  - ESC-Taste und Windows(X)-Button Code Handling (Erkläre ich alles später!)

Keyboard Handling



- Die Abfrage, ob im Moment eine bestimmte Taste gedrückt ist, ist einfach:
- **if (Keyboard.*isKeyDown(Keyboard.KEY_ESCAPE))***

- *Entsprechend für andere Tasten, nutzt die Codecompletion von Eclipse…*

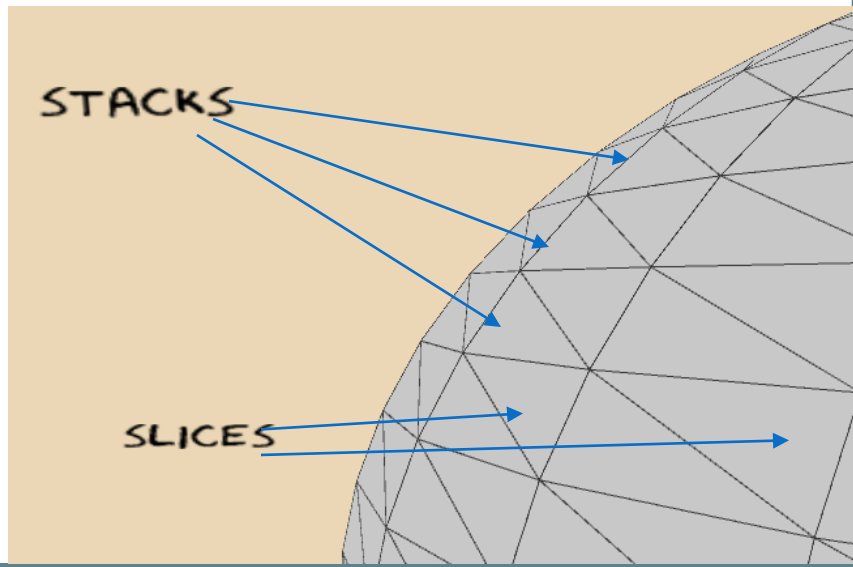
Aufgabe 1 - Die Kugel

1. Zeichnen Sie eine **gelbe Kug**el in der Mitte des Bildschirms. Benutzen Sie die Methode **GLDrawHelper.drawSphere()** um eine Kugel zu zeichnen.
2. Programmieren Sie dass durch **die SPACE Taste** zwischen **Wireframe** und **(normaler)** gefüllter Darstellung hin- und her geschaltet wird:
   - GL11.*glPolygonMode(GL11.GL_FRONT_AND_BACK, GL11.**GL_LINE**);*
   - GL11.*glPolygonMode(GL11.GL_FRONT_AND_BACK, GL11.**GL_FILL**);*
3. *Machen Sie die Sphere 200 Einheiten groß und Golfball-groß auf dem Bildschirm (Hint: USE lookat() um die Kamera „weiter hinten“ zu platzieren)*

Draw a Sphere

- It is simple to create a sphere
- Integrate GLDrawHelper.java in your Project
- GLDrawHelper.**drawSphere(float radius, int slices, int stacks);**

  - **Sphere of Radius radius**
  - **At Coordinates (0,0,0)**
  - **Actual draw color used**
  - **e.g. 12,12 for slices,stacks**



Objekte malen

- Nun können wir unsere Objekte malen.
- Um nicht alle am Ursprung zu platzieren können wir
- GL11.*glTranslatef(x, y, z);*
- GL11.*glRotatef(angle, x, y, z); //angle in degrees!*
- GL11.*glScalef(x, y, z);*
- benutzen, um Objekte (bzw. das Koordinatensystem) zu verschieben.
- Damit können wir unsere Sphere überall und in jeder Größe platzieren!

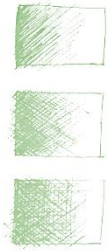
Objekte malen

- Natürlich können wir unsere Objekte auch direkt über Vektoren bestimmen (die aber relativ zur momentanen ModelView Matrix sind)

```
gl.glBegin(GL2.GL_TRIANGLES);
    gl.glColor3f(1.0f, 0.0f, 0.0f);
    gl.glVertex3f(10f, 10f, -100.0f);
    gl.glColor3f(0.0f, 1.0f, 0.0f);
    gl.glVertex3f(25f, 35f, -100.0f);
    gl.glColor3f(0.0f, 0.0f, 1.0f);
    gl.glVertex3f(40f, 10f, -100.0f);
    gl.glEnd();
```

Aber das würde jegliche Optimierung für die GraKa unterlaufen und die Wiederverwendbarkeit des Codes schmälern.

Reihenfolge



- Absolut wichtig:
- **Bei OpenGL wird mit jeder Operation das Koordinatensystem verändert, folgende (Mal-) Operationen beziehen sich dann auf das veränderte Koordinantensysten (KOOSys)!**
- Das heißt, die Veränderungen beziehen sich nicht immer auf ein gedachtes KOOSys mit den kanonischen XYZ-Achsen!

Beispiel

- // Rotation around mother
- GL11.gl**Rotate**f(**mRot**, 0.f, 1.f, 0.f);
- GL11.gl**Translate**f(**motherDistance**, 0, 0);
- // own rotation
- GL11.gl**Rotate**f(**eRot**, 0, 1, 0);
- **drawSphere**(…); //passiert am jetzigen KOO-Ursprung

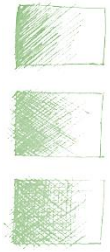
Alternative Lesart von glVertex nach oben:

- → Drehe den Planeten/Kugel um sich selber mit ***eRot***
- → Verschiebe um **motherDistance** aus dem Ursprung
- → Drehe ihn um den Ursprung mit ***mRot***

Aufgabe 2

- Malen Sie die Sonne (gelber Ball in der Mitte) mit einer Erde, die sich um die Sonne dreht!
  - Sonne steht fest
  - Planet Erde rotiert um die Sonne (in 356 Tagen)
  - Planet Erde rotiert um sich selbst (in 24h == 1Tag)
- Hmm, um zu sehen, ob das richtig implementiert ist muss ich ein Jahr auf den Bildschirm schauen…
- Das ist mir zu lang ;-)
- → **Die Zeit soll als 10 Tage pro Sekunde simuliert werden!**

Echte Zeit, simulierte Zeit



- Wir brauche eine Methode, die die echte vergangene Zeit misst und uns die simulierte vergangene Zeit zurückliefert!

- public float getSimDay();
  - liefert den **aktuell simulierten Tag** als float
  - startet mit **0**
  - ist nach einer **1/100stel** Sekunde bei 0,1
  - ist nach **einer** Sekunde bei **10**
  - nutzt die Variable **float days_per_second = 10**

- **Zeit in Sekunden (seit dem 1.1.1970):**
- **(float)(Sys.*getTime() * 1.0) / Sys.getTimerResolution();***
  - *merkt Euch die Startzeit der App, die vergangene Zeit zu berechnen*

Problem

- Die Erde ist **immer vor/hinter der Sonne**!
- OpenGL zeichnet die Daten immer in der Reihenfolge, wie sie geliefert werden.

- Lösung: **DEPTH_BUFFER einschalten**:
  Dann werden die Pixel der Polygone entsprechend Ihrer **Entfernung zur Kamera** gezeichnet, wie man es erwartet:
  - **GL11.glEnable(GL11.*GL_DEPTH_TEST);*** *// in setup()*

- *Beseitigen der gesetzten z-Buffer bits // in draw(), wie gewohnt*
- **GL11.glClear(GL11.*GL_COLOR_BUFFER_BIT | GL11.GL_DEPTH_BUFFER_BIT)***

```
GL11.glClear(GL11.GL_COLOR_BUFFER_BIT | GL11.GL_DEPTH_BUFFER_BIT);
```

Sonne, Mond und Sterne

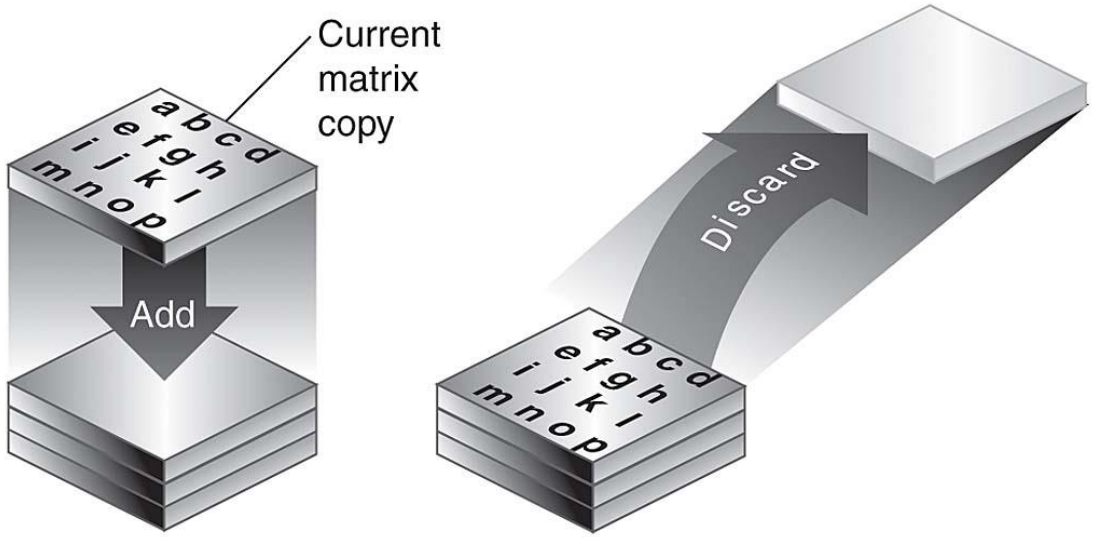
- Sonnensystem:
- Sonne (109 mal Erdgröße, wir nehmen weniger! E.g. 10-mal)
- Planeten: Merkur Venus Erde Mars (Jupiter Saturn Uranus Neptun) (Größen anpassen)
- Monde: Bitte nur Erde und Mars, sonst wird es zu aufwendig. Mond vielleicht 0.3 mal die Größe des Mutterplaneten
- Umlaufzeiten: Bitte wie Tabelle
- Umlaufbahn: Bitte als einfache Kreisbahn.
- In ROT: Änderungen von der Realität
- Daten: hier

Aufgabe 3

- Bauen Sie einen Sonnensystem-Simulator!
- Planeten rotieren um die Sonne!
- Monde rotieren um die Planeten (v=??).
- Planeten rotieren um sich selbst.
- Beobachter steht irgendwo außerhalb.
- Größenverhältnisse stimmen (bis auf Sonne)
- Geschwindigkeiten stimmen.

Die aktuelle Matrix sichern / wiederherstellen

- glPushMatrix() / glPopMatrix():
- Push the actual matrix onto a stack
- Retrieve it from the stack
- Helpful functions for dealing with hierarchical models…

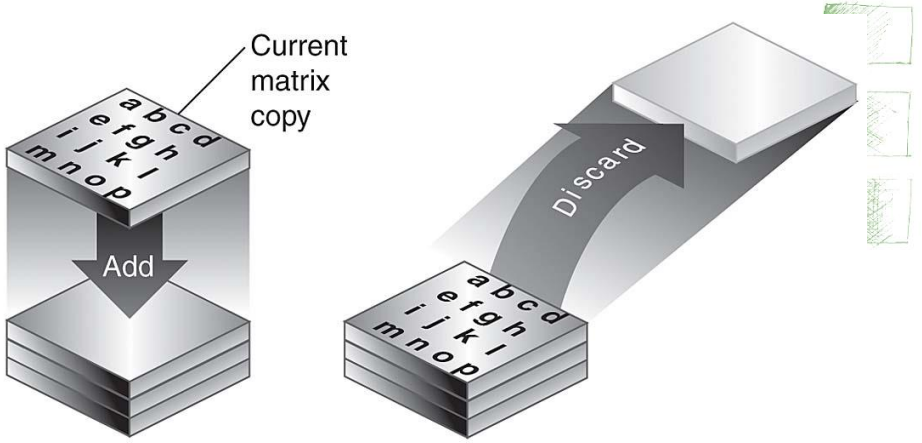


Info and images taken from Shreiner: OpenGL Programming and here

Example

- Car wheel with bolts

```
draw_wheel_and_bolts()
{
  int i;

  draw_wheel();
  for(i=0;i<5;i++){
   glPushMatrix();
     glRotatef(72.0*i, 0.0, 0.0, 1.0);
     glTranslatef(3.0, 0.0, 0.0);
     draw_bolt();
   glPopMatrix();
  }
}
```




```
draw_body_and_wheel_and_bolts()
{
  draw_car_body();
  glPushMatrix();
              /*move to first wheel position*/
   glTranslatef(40, 0, 30);
   draw_wheel_and_bolts();
  glPopMatrix();
  glPushMatrix();
             /*move to 2nd wheel position*/
   glTranslatef(40, 0, -30);
   draw_wheel_and_bolts();
  glPopMatrix();
  ...          /*draw last two wheels
similarly*/
}
```

Rangehensweise - Vorschläge

- Sonne ist FIX. Vielleicht an 0,0,0?
- Planet == Himmelskörper (also auch Sonne und Monde):
  - Größe,
  - Eigenrotation (speed),
  - Mutterplanetumlauf (speed),
  - Abstand Mutter,
  - Aktuelle Rotation (2x)
  - **Liste Kinder: List<Planet>**
- Monde: isA Planet…

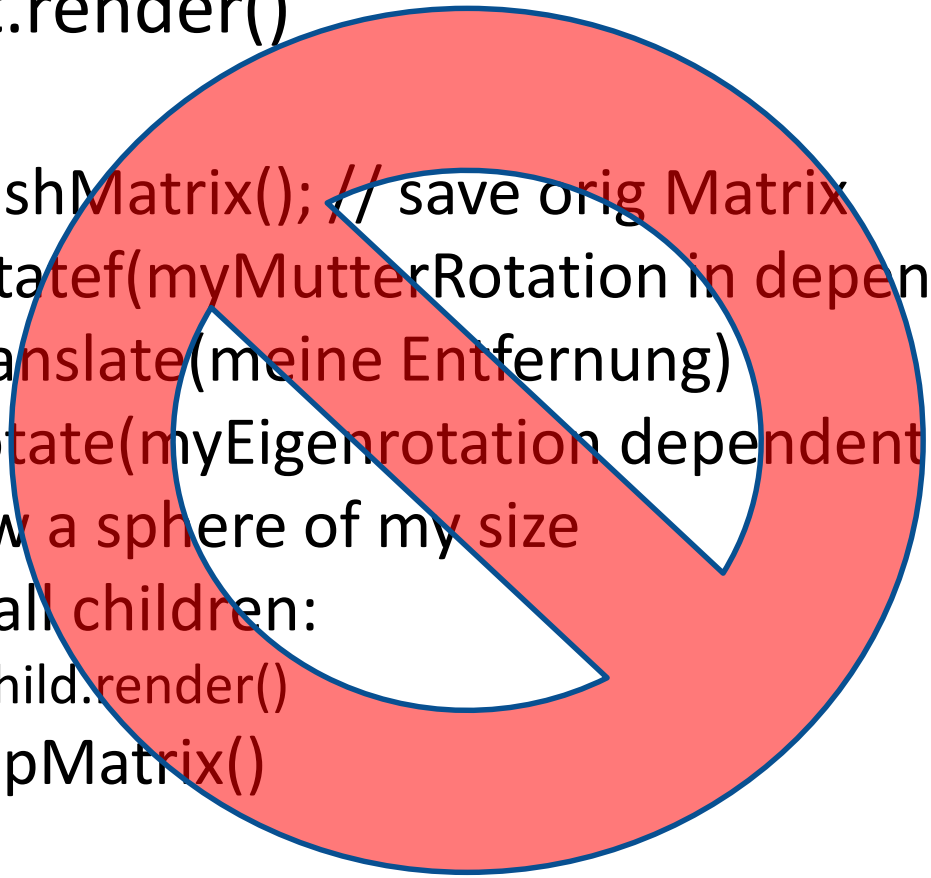
- Alle Planeten:
- Eigene render() Methode und update() Methode
  - Alle: in render und update: **erst für mich, dann für alle Kinder von mir.**

Hierarchischer Baum

- Bitte beachten Sie:
  - **Die Sonne, Planeten und Monde bilden eine hierarchische Baumstruktur!**
  - So soll das ganze auch in der **Umsetzung** sein.
- Das heißt:
- Das Hauptprogramm sagt ausschließlich:
  - **sun.update() und sun.draw()**
  - Das update/draw() aller anderen wird über eine Baumhierarchie ausgelöst
  - Sonne → Planeten → Monde

Planet render() Methode

- Planet.render()
- {
  - glPushMatrix(); // save orig Matrix
  - glrotatef(myMutterRotation in dependance of time);
  - glTranslate(meine Entfernung)
  - glRotate(myEigenrotation dependent on time)
  - Draw a sphere of my size
  - For all children:
    - child.render()
  - glPopMatrix()
  - }



Planet draw() Methode

- **Planet.draw()**
- {
  - glPushMatrix(); // save orig Matrix
  - glrotatef(myMutterRotation in dependance of time);
  - glTranslate(meine Entfernung)
  - For all children:
  - **child.draw()**
  - glRotate(myEigenrotation dependent on time)
  - Draw a sphere of my size
  - glPopMatrix()
  - }

Sonst würde der Mond jeden Tag um die Erde kreisen und jeden Monat nochmal extra… ist **abweichend** von einem normalen hierarchischen System!

Verbesserungen

- Überprüfen der Rotation mit Wireframe
  - Einfacher zu sehen!
- Und verschieden Farben
  - Bauen Sie für die Planets noch eine Color ein.
    - Float r,g,b;
    - Und vor dem Malen des Planeten
    - GL11.glColor3f(color_r, color_g, color_b);

Aufgabe 4

- Besonders 3D sieht das nicht aus.
- Es fehlt Licht und eine glaubhafte Schattierung der Planeten.

- Bauen Sie eine leuchtende Sonne!

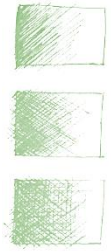
Lights on!

- Erst mal eine Lichtquelle definieren:

- **Mögliche Werte:**
- FloatBuffer noAmbient = GLDrawHelper.*directFloatBuffer(**new float[] {00.0f, 0.0f, 0.0f, 1.0f});*
- FloatBuffer whiteDiffuse = GLDrawHelper.*directFloatBuffer(**new float[]{1.0f, 1.0f, 1.0f, 1.0f});*
- FloatBuffer position = GLDrawHelper.*directFloatBuffer(**new float[]{0.0f, 0.0f, 0.0f, 1.0f});*

-

Light on – Create a light.



- Nun noch die Werte an OpenGL übergeben und das Licht anknipsen.
- Dazu in init():
  - GL11.glEnable(GL11.GL_LIGHTING);
  - GL11.glLight(GL11.*GL_LIGHT0, GL11.GL_AMBIENT, noAmbient);*
  - GL11.glLight(GL11.*GL_LIGHT0, GL11.GL_DIFFUSE, whiteDiffuse);*
  - GL11.glEnable(GL11.*GL_LIGHT0);*
- In draw():
  - GL11.glLight(GL11.*GL_LIGHT0, GL11.GL_POSITION, position);*
  - *Muss ich in draw() machen, **nach** dem gluLookAt()*

Bring back the color

- Nichts als Ärger, jetzt sind die FARBEN weg….
- Mit GL_LIGHTING haben wir uns für „Materials“ statt „Colors“ entschieden.
- Erfreulicherweise können wir OpenGL aber sagen: Benutze unsere Colors als (Default-) Material:
- GL11.glColorMaterial (GL11.*GL_FRONT_AND_BACK, GL11.GL_AMBIENT_AND_DIFFUSE) ;*
- GL11.glEnable (GL11.*GL_COLOR_MATERIAL) ;*
- → *auch in setup()*