Computergrafik

Universität Osnabrück, Henning Wenke, 2012-05-15

Noch Kapitel V:

Modeling Transformation & Vertex Shader

Vertex Shader Fragen I

```
#version 330 core

in vec4 positionMC;
uniform mat4 modelCoords_2_worldCoords_Transform;

void main(){

  vec4 positionWC = modelCoords_2_worldCoords_Transform * positionMC;

}
```

- Was berechnet dieser Shader?
  - Nichts. Keine out-Variablen geschrieben
- Enthält Vertex Shader `out`-Variablen?
  - Ja, u. a. die Build-In Variable `gl_Position`
- Kann man dem Code ansehen, dass es sich um einen VS handelt?
  - Nein. Keine VS-spezifischen Build-In Variablen verwendet

Vertex Shader Fragen II

```
#version 330 core

uniform mat4 modelCoords_2_worldCoords_Transform;

void main(){

  vec4 positionMC = vec4(…, 1.0);
  gl_Position = modelCoords_2_worldCoords_Transform * positionMC;

}
```

- Was berechnet dieser Shader?
  - Eine transformierte Koordinate…
  - … die für alle Instanzen des VS gleich ist…
  - … da keine in-Variablen verwendet werden

Vertex Shader Fragen III

```
#version 330 core

in float angle;
in vec4 positionMC;

void main(){

  mat4 rotZ = mat4( cos(angle), sin(angle), 0.0, 0.0,  // Column 0
                   -sin(angle), cos(angle), 0.0, 0.0,  // Column 1
                    0.0,        0.0,        1.0, 0.0,  // Column 2
                    0.0,        0.0,        0.0, 1.0   // Column 3
                );
  gl_Position = rotZ * positionMC;
}
```

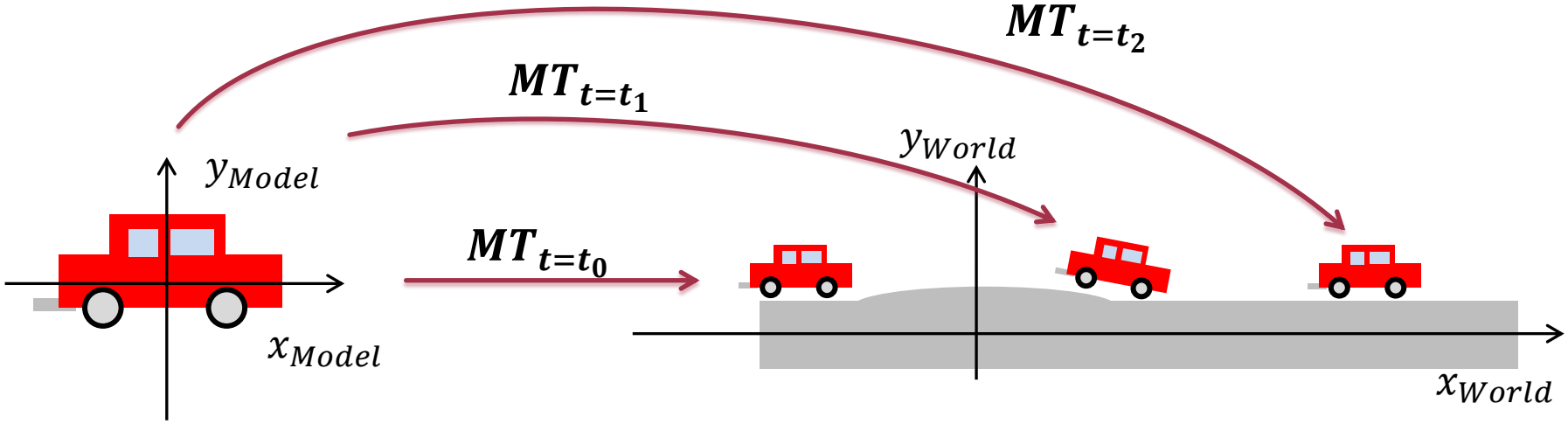
- Was berechnet dieser Shader?
  - Eine um die z-Achse rotierte Koordinate
  - Jeder Vertex wird anders rotiert, da Matrix in Abhängigkeit einer in-Variablen
  - Ausgangsgeometrie wird völlig verformt

Vertex Shader Fragen IV

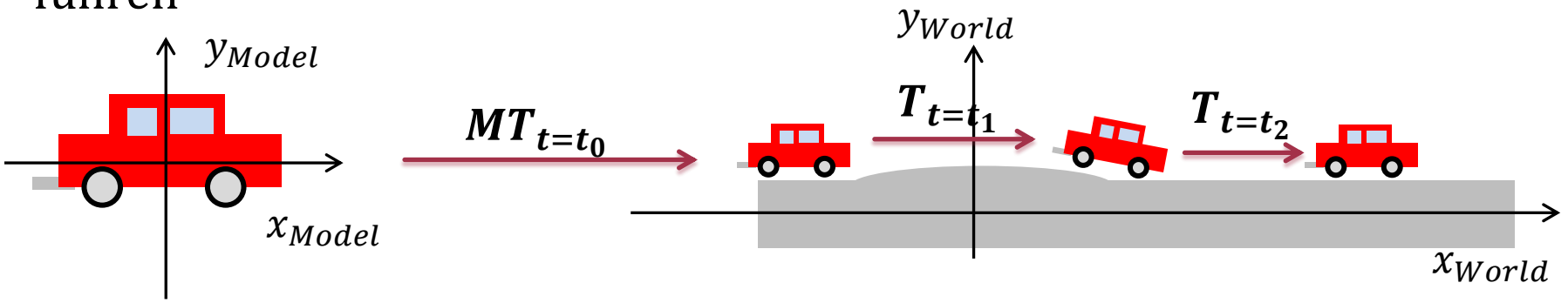
```
#version 330 core

in float angle;
in vec4 positionMC;

void main(){

gl_Position = vec4(
                positionMC.x * cos(angle) - positionMC.y * sin(angle),
                positionMC.x * sin(angle) + positionMC.y * cos(angle),
                positionMC.z,
                positionMC.w
              );
}
```

- Was berechnet dieser Shader?
  - Mit Shader der letzten Folie identisches Ergebnis, aber:
  - 4 Multiplikationen, 2 Additionen, statt:
  - 16 Multiplikationen, 12 Additionen + 16 float zusätzlichen Speicher

Vertex Shader Fragen V (a)



- Alternative zur der Modeling Transformation bei Animation:
- Verändere Position des Objekts selbst
- Kann nach einigen Transformationen zu Genauigkeitsproblemen führen



Vertex Shader Fragen V (b)

- VS zum Manipulieren der original Vertex Daten möglich?
- Originaldaten unüberschreibbar, in-Variablen lediglich lokale Kopien
- Lösungsansätze
  - Transform Feedback
  - OpenCL
  - Später mehr

```
#version 330 core

in vec4 oldPosition;
uniform mat4 updatePosMatrix;
out vec4 newPosition;

// Vertex Shader, geeignet zum Einsatz mit Transform Feedback
void main(){
  newPosition = updatePosMatrix * newPosition; // nicht: glPosition
}
```

Vertex Shader Fragen VI

- Ein Objekt soll an verschiedenen Stellen in der Szene platziert werden
- Kann ein Aufruf der Graphics Pipeline mit dem VS unten dies leisten?
  - Nein
- Vorgehen?
- Für alle Instanzen i des Objekts:
  - Setze Matrix für Modeling Transformation für Instanz i
  - Wende VS auf alle Vertices der Instanz i an

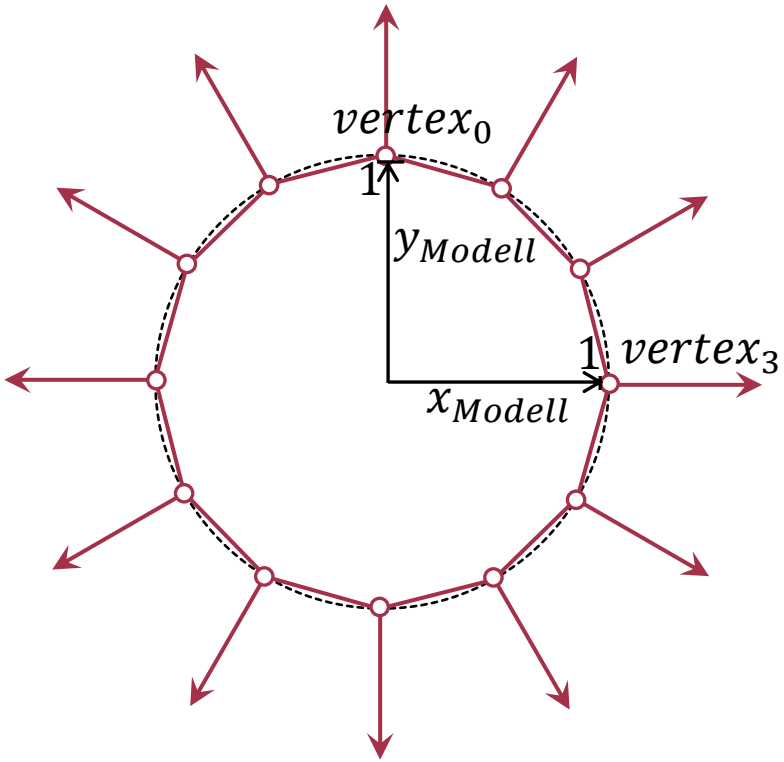
```
#version 330 core

in vec4 posMC;
uniform mat4 modelingTransform;

// Vertex Shader für Modeling Transform
void main(){
  glPosition = modelingTransform * posMC;
}
```

5.5

Beispiel II: Translation, Rotation und einfache Beleuchtung einer Geometrie im VS

Beispielmodell eines Kreises

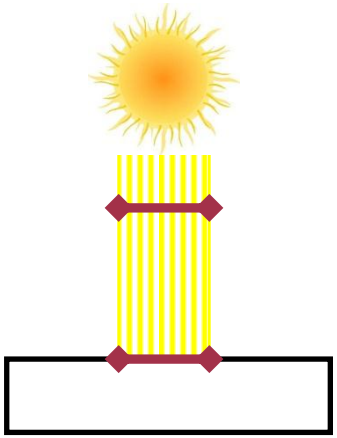
$$posMC_0 = \begin{pmatrix} 0 \\ 1 \\ const \\ 1 \end{pmatrix} \qquad normalMC_0 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

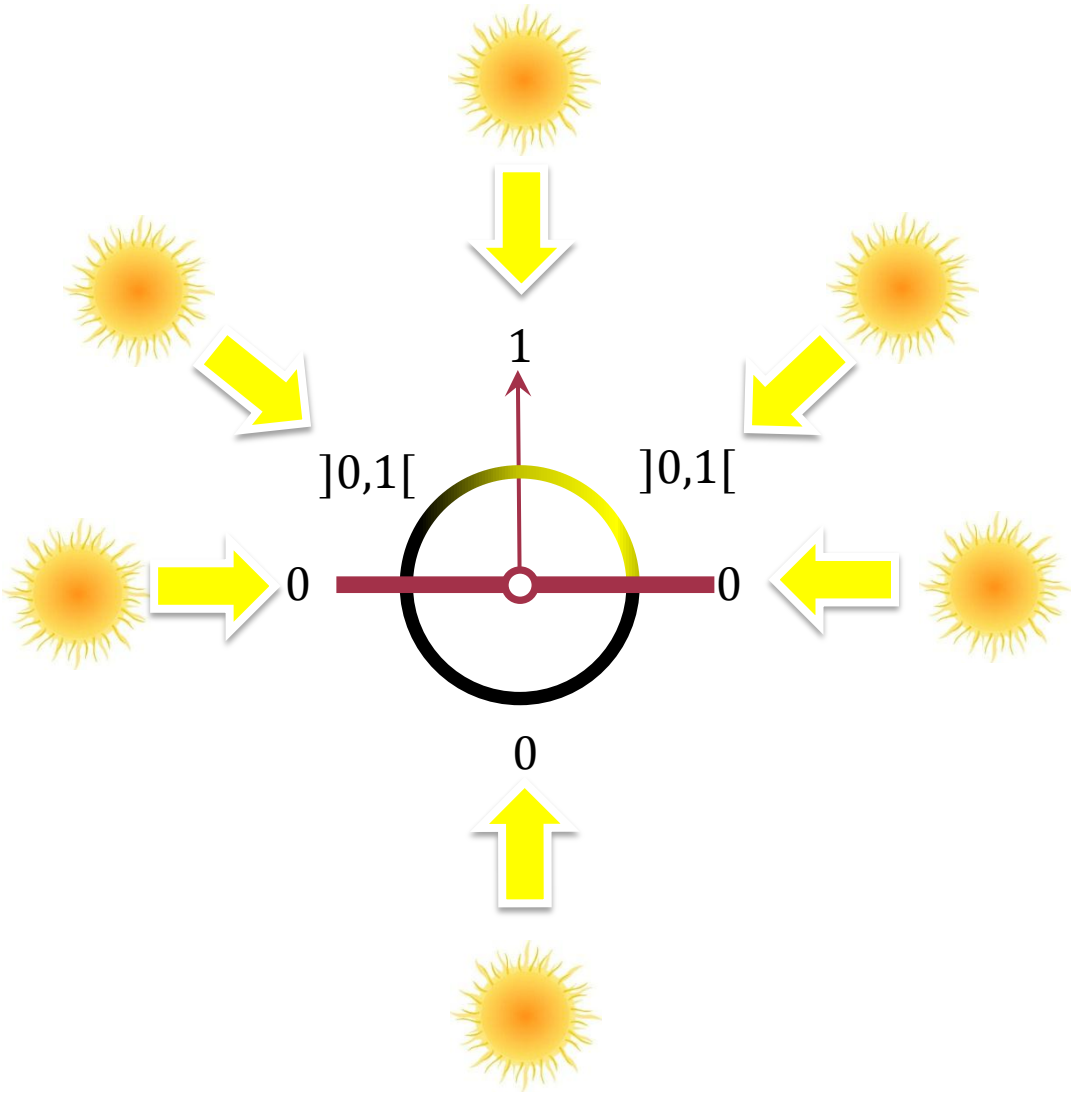


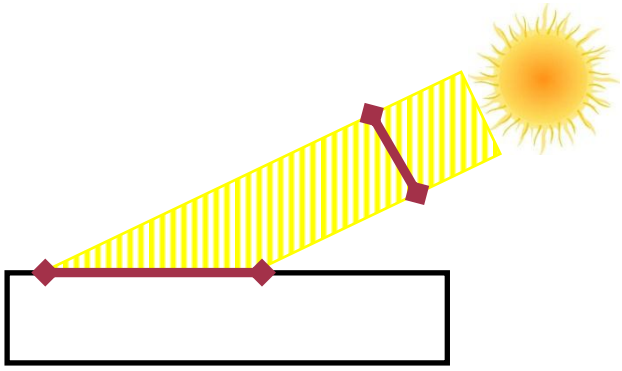
$$posMC_3 = \begin{pmatrix} 1 \\ 0 \\ const \\ 1 \end{pmatrix} \qquad normalMC_3 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

- Gegeben: Modell, hier: Kreis in xy-Ebene
- Genähert durch 12 Vertices
- Jeder Vertex besteht aus:
  - Position "`posMC`" $(px_i, py_i, const, 1)$ in Modellkoordinaten
  - Oberflächennormale "`normalMC`" $(nx_i, ny_i, 0)$ in Modellkoordinaten

Exkurs: Lighting (qualitativ) I



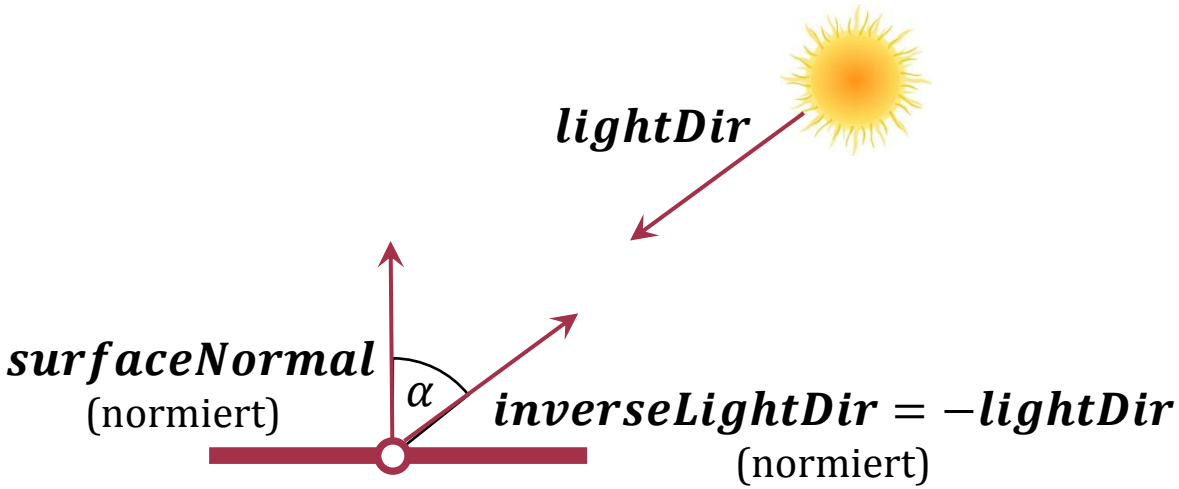


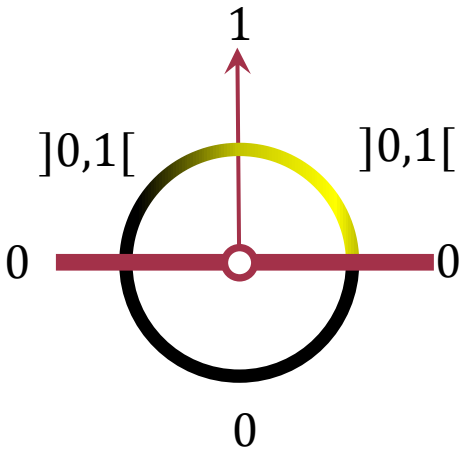


Exkurs: (qualitativ) Lighting II

- Ziel: Paralleles Licht aus Lichtquelle im Unendlichen
- Gegeben:
  - Vertices mit Normale in Weltkoordinaten
  - Lichtrichtung
- Gesuchte Funktion: $\sim cos$, oder:
- $brightness = max(dot(\boldsymbol{inverseLightDir},\ \boldsymbol{surfaceNormal}), 0)$;

$$\cos(\alpha) = \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{|\boldsymbol{a}||\boldsymbol{b}|}$$





Vertex Shader & Daten

- Vertex Attribute vor VS Ausführung
  - Position in Model Coords: "`posMC`"
  - Normale in Model Coords: "`normalMC`"
- Ziel:
  - Geometrie rotiert um eigene Achse…
  - … und wird in Szene verschoben (Modeling T)
  - Vertices werden beleuchtet
- Vertex Attribute nach VS Ausführung
  - Position in World Coords: "`gl_Position`"
  - Brightness: "`brightness`"

Modeling Transform & Lighting VS

```
#version 330 core
// posMC, normalMC
in vec4 posMC;      // Position in Model Coordinates
in vec3 normalMC;   // Oberflächennormale in Model Coords (normiert)
// rotate_z, translate, inverseLightDir
uniform mat4 rotate_z, translate;          // Transformationsmatrizen
uniform vec3 inverseLightDir;              // -Lichtrichtung (normiert)
// brightness
out float brightness;

void main(){
  // Modeling Transformation der Position
  gl_Position = translate * rotate_z * posMC;

  // Modeling Transformion der Normalen
  vec3 normalWC = mat3(rotate_z) * normalMC; // mit upper-Left 3x3 Mat

  // Simple Vertex Lighting
  brightness = max(dot(normalWC, inverseLightDir), 0);
}
```

MT & L Vertex Shader (simplified)

```
#version 330 core
in vec3 normalMC;
in vec4 posMC;

// Vorberechnete Transformationsmatrix für Position:
// mc2wc_Pos = translate * rotate_z;
uniform mat4 mc2wc_Pos;

// Vorberechnete Transformationsmatrix für Normale:
// mc2wc_Normal obere linke 3x3 Matrix der Rotationsmatrix;
uniform mat3 mc2wc_Normal;
uniform vec3 inverseLightDir;

out float brightness;

void main(){
  gl_Position    = mc2wc_Pos * posMC;
  vec3 normalWC = mc2wc_Normal * normalMC;
  brightness = max(dot(normalWC, inverseLightDir), 0);
}
```

5.6

Spezifikation & Anbinden der Uniform Daten

LWJGL Praxis: Matrix aufstellen

```
// Erzeugt org.lwjgl.util.Matrix4f. Initialisiert als Einheitsmatrix
Matrix4f mat = new Matrix4f(); Matrix4f mat2 = new Matrix4f();
Matrix4f mc2wc_Pos_Java = new Matrix4f();

// Erzeugen eines org.lwjgl.util.Vektors3f. Repräsentiert gew. Translation
Vector3f tx = new Vector3f(1, 0, 0);

// Erzeugen eines Vektors3f. Repräsentiert Rotationsachse (z-Achse)
Vector3f zAxis = new Vector3f(0, 0, 1);

// Berechnet:  mat = mat * Translation(tx.x, tx.y, tx.z).
// mat ist anschließend Matrix für Translation in x-Richtung um 1
mat.translate(tx);

// Setzt mat2 als Matrix zur Rotation um die z-Achse um 180 Grad.
mat2.rotate(Math.PI, zAxis);

// Liefert unsere gesuchte Matrix für Rotation um z-Achse mit anschließender
// Translation.
mc2wc_Pos_Java.translate(tx).rotate(Math.PI, zAxis);
```

LWJGL Praxis: Matrix als FloatBuffer

```
// Erzeugen eine Instanz der Java-Klasse java.nio.FloatBuffer mit einer
// LWJGL-Hilfsfunktion. Liefert einen size-elementigen FloatBuffer
java.nio.FloatBuffer org.lwjgl.BufferUtils.createFloatBuffer(int size);
```

```
// Erzeugen eines (leeren) FloatBuffers zur Datenübergabe an OpenGL
FloatBuffer mc2wc_Pos_Buffer = createFloatBuffer(16);

// Speichert Werte der Matrix "mc2wc_Pos_Java" der letzten Folie im FloatBuffer
// "mc2wc_Pos_Buffe" in column-major-order.
// Setzt anschließend mc2wc_Pos_Buffer.position auf 16
mc2wc_Pos_Java.store(mc2wc_Pos_Buffer);

// Setze mc2wc_Pos_Buffer.position auf 0, da ab hier die Werte gelesen werden…
mc2wc_Pos_Buffer.position(0)
```

```
// Jetzt das Ganze noch für "mc2wc_Normal_Buffer"!
// …
```

Ausgangslage

- Wir haben:
- Shader, hier: Vertex Shader Folie 16
- Uniform Daten. Hier: unsere Matrizen, in den FloatBuffern
  - "`mc2wc_Pos_Buffer`"
  - "`mc2wc_Normal_Buffer`"
  - (letzte Folie)
- Ein gelinktes Program Object, "`circlePO`", beeinhaltend unseren VS

```
#version 330 core
// in: nächste Woche
in vec3 normalMC;
in vec4 posMC;

uniform mat4 mc2wc_Pos;
uniform mat3 mc2wc_Normal;
uniform vec3 inverseLightDir;

// Rest uninteressant
out float brightness;

void main(){
 …
}
```

Anbinden der Uniform Daten I

- OpenGL vergibt für jede Uniform-Variable beim Linken des Program Objects einen Index
- Muss geholt werden, um Variable anzusprechen

```
// Liefert den Index einer Uniform Variable
// Eindeutig nur für dieses Program Object
int glGetUniformLocation(
  int program, // gelinktes PO, welches die Shader enthält
  String name  // Name der Uniform Variable
);
```

```
// Hole & speichere Indices unserer Uniform Variablen "mc2wc_Pos" und
// "mc2wc_Normal" unseres PO "circlePO"
int index_MT_Pos    = glGetUniformLocation(circlePO, mc2wc_Pos);
int index_MT_Normal = glGetUniformLocation(circlePO, mc2wc_Normal);
```

Anbinden der Uniform Daten II

- Mit glUniform[...] können Daten an Uniform Variablen angebunden werden

```
glUniform<Angaben über Parameter>(); // Letzte Woche: glUniform4fv
```

- Dazu muss das PO als Teil des GL-State gesetzt sein

```
// Setzt Uniform Variable, hier 4x4 Matrix, des gerade aktiven PO
glUniformMatrix4(
  int location,       // Index der Uniform Variable
  boolean transpose, // Soll Matrix transponiert werden ?
  java.nio.Buffer    // Die Daten, siehe Folie 20
); // Hinweis: In "C-GL": glUniformMatrix4fv(...);
```

```
// Zuerst: Aktiviere PO, dessen Uniforms zu setzen sind
glUseProgram(circlePO);

// Binde dann unsere Daten (Folie 20) an
glUniformMatrix4(index_MT_Pos,    false, mc2wc_Pos_Buffer);
glUniformMatrix4(index_MT_Normal, false, mc2wc_Normal_Buffer);
```

Ausblick: Fehlendes

- Spezifikation & Übergabe der Vertex Daten
- Weitere Berechnungen im VS
  - 3D-Szene, freie Kamera
  - Perspektive
- Weiterer Verlauf der Pipeline