Computergrafik

Universität Osnabrück, Henning Wenke, 2012-05-14

Kapitel V:

Modeling Transformation & Vertex Shader

5.1

Vertex

Definitionen: Vertex

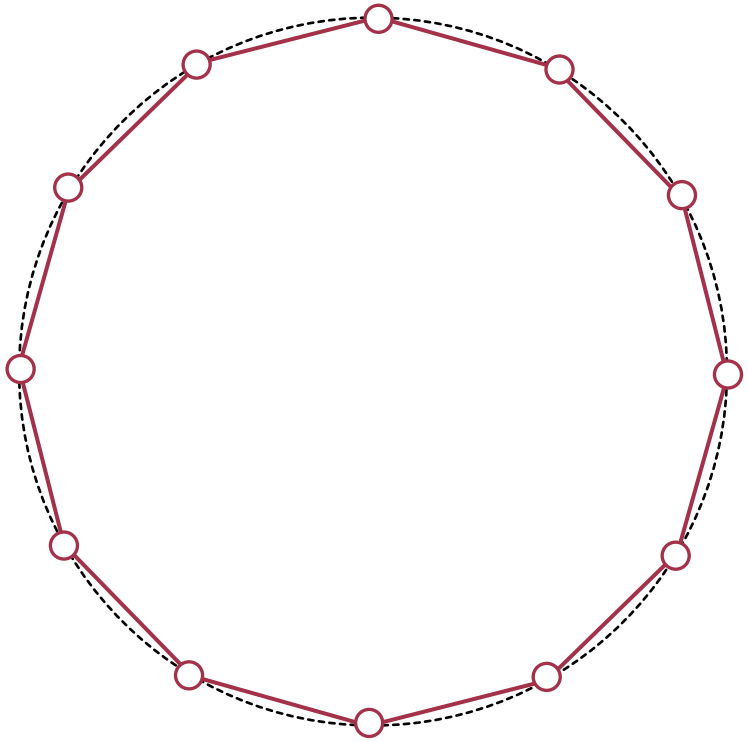
Vertex Computergrafik ≔ Mathematischer Punkt auf einer Oberfläche zusammen mit Eigenschaften an dieser Stelle

Oft “Eckpunkt“ eines Primitives (geometrische Figur)

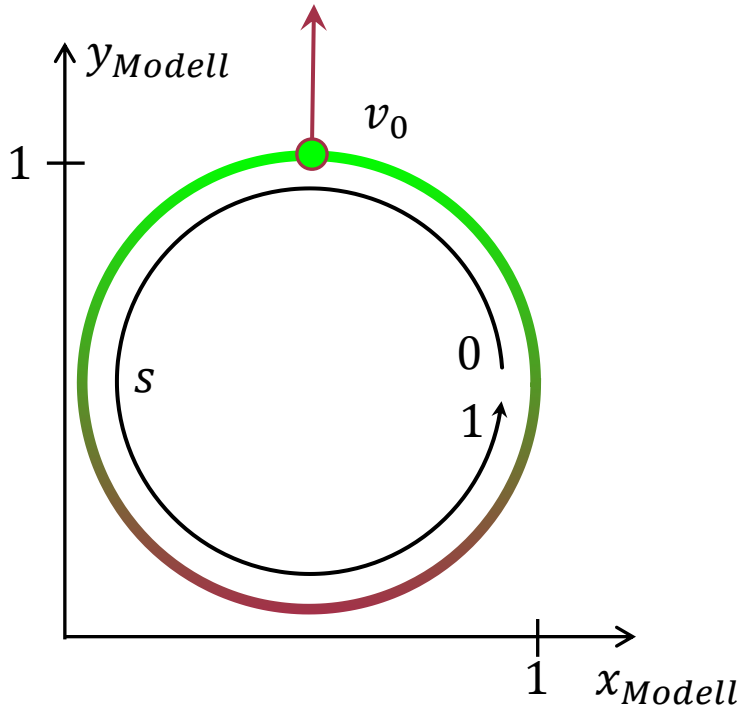
Vertex OpenGL ≔ Datenstruktur. Geeignet, um einen Vertex (Computergrafik) zu repräsentieren

Vertex als Oberflächenpunkt

- Gegeben: Kontinuierliches Modell (Volumen)
- Gesucht: Diskrete Näherung durch sinnvolle Verteilung von Vertices auf dessen Oberfläche
- Enthalten Eigenschaften der Stelle
- Sind dann typischerweise Eckpunkte geometrischer Objekte, welche Oberfläche annähern
- Dazu zusätzlich Topologie nötig:
  - Vorschrift zum Zusammenfügen der Vertices zu Dreiecken, Linien, etc.
  - Später…



Typische Vertex Attribute



- Kann v. A. geometrische Informationen enthalten, z.B.:
  - Position in einem KS (fast immer)
  - Normale in einem KS
  - Oberflächenparameter, hier: Linienparameter
- Auch beliebige andere Informationen, z.B.:
  - Farbe
  - Deformierbarkeit
  - …

Attribute v_0:

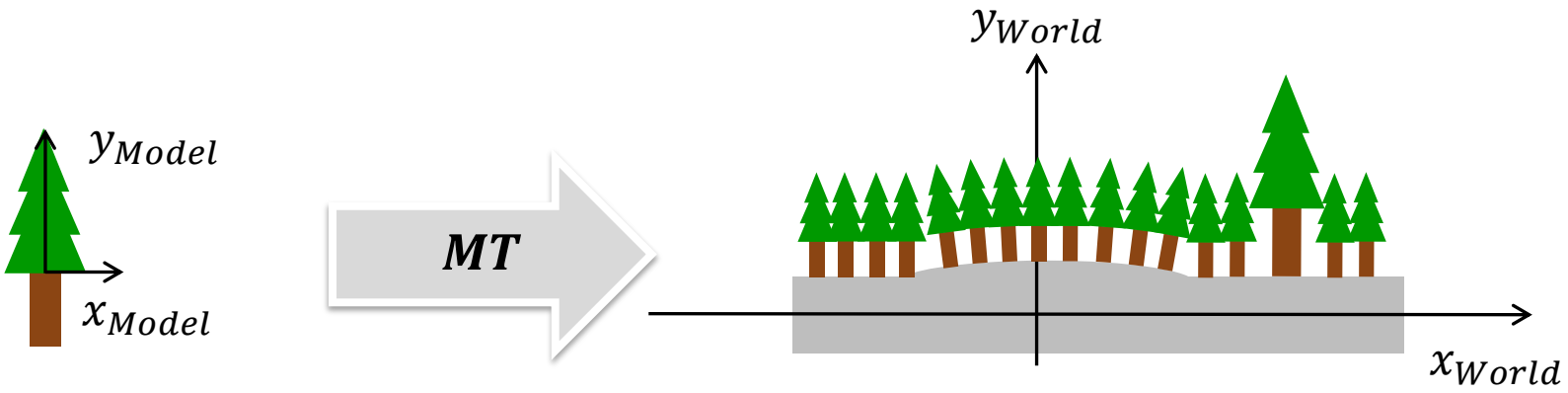
- Position: (0.5, 1.0)
- Normale: (0.0, 1.0) ,(normiert)
- Linienparameter s: 0.25
- Farbe: grün

5.2

Modeling Transformation

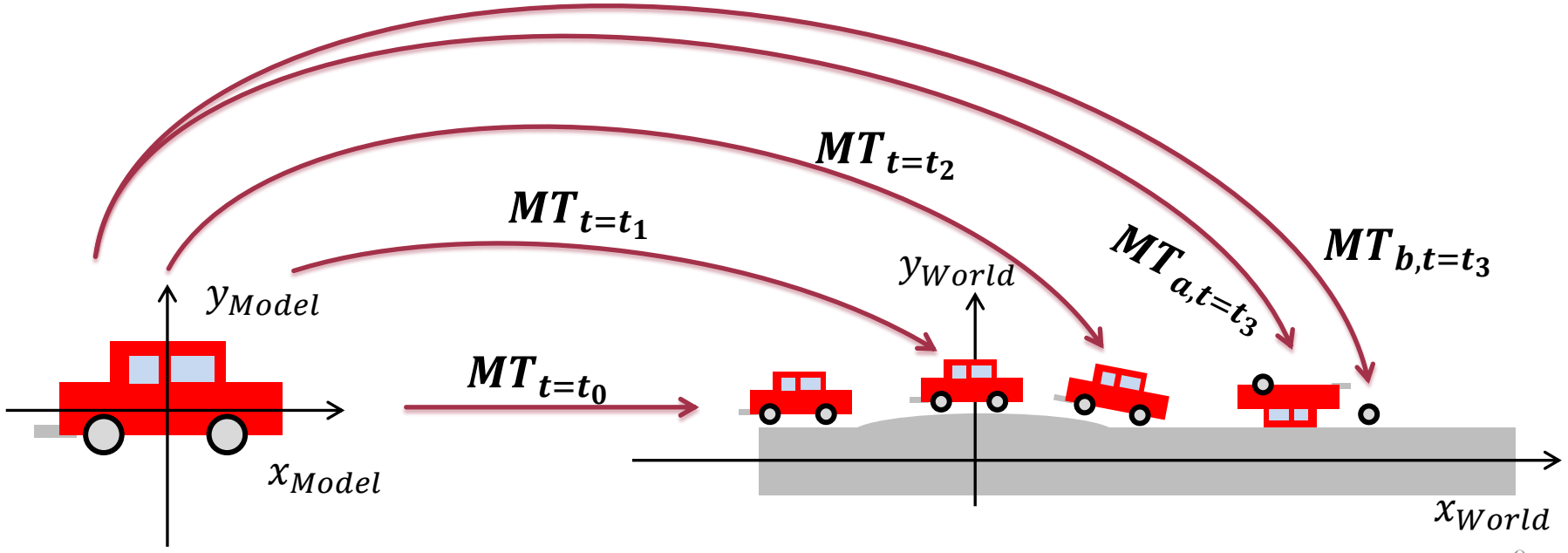
Modeling Transformation I

- Objekte i.d.R. in eigenem KS modelliert: “Model Coordinates“
- Platzierung in Szene durch Modeling Transformations
  - Translation
  - Rotation
  - Skalierung, …
- Räumliche Wiederverwendung der Objekte möglich
- Sind dann in globalem KS, den “World Coordinates“ beschrieben



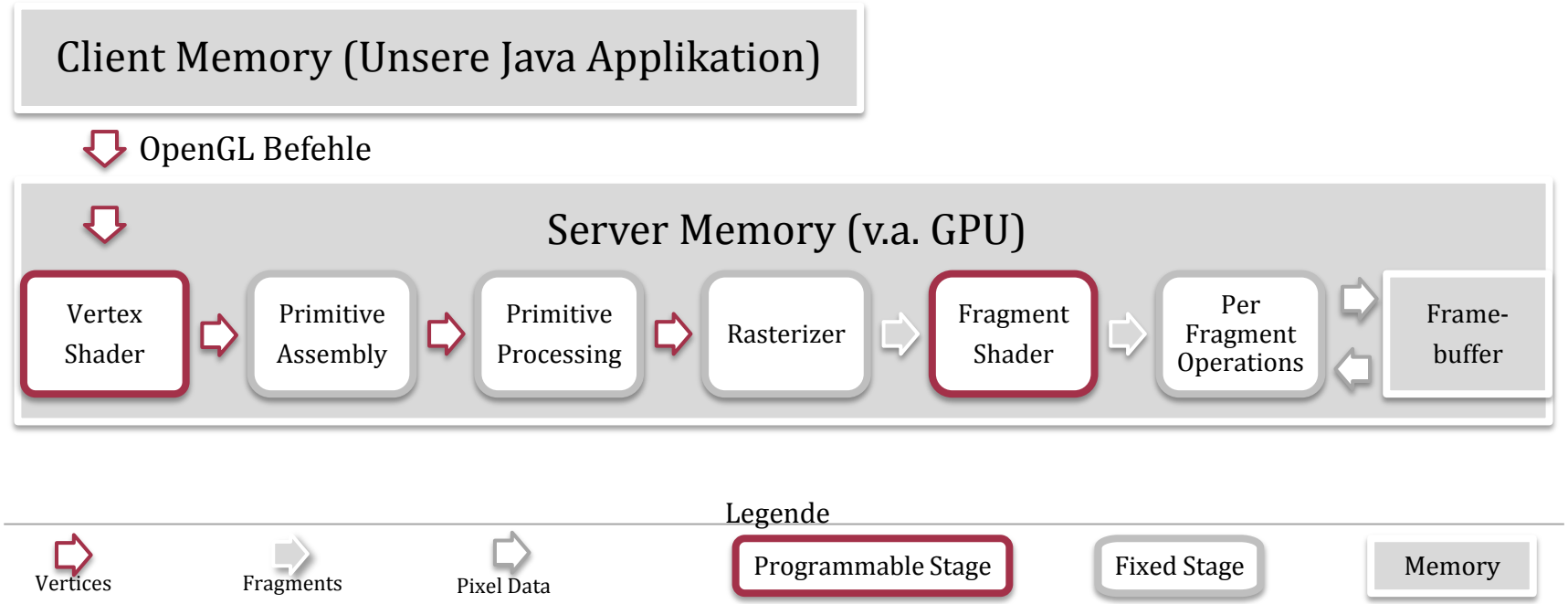
Modeling Transformation II

- Zeitliche Wiederverwendung der Objekte möglich
- Typisch: Anwendung verschiedener Matrizen auf jeweils alle Vertices einzelner Teile des Objekts
- Alternativ: Szenenangabe direkt in *World Coordinates*
- *Modeling Transformation* dann überflüssig



Modeling Transformation III

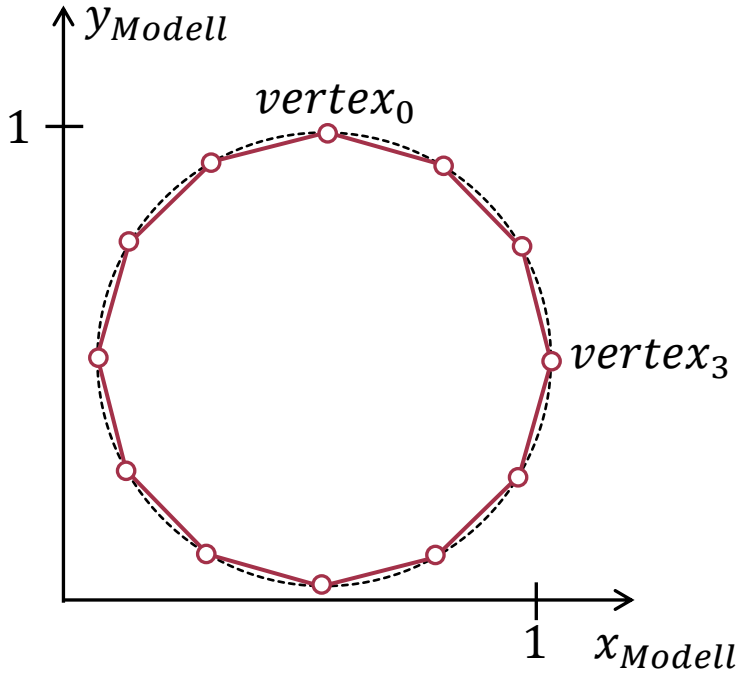
- Modeling Transformation kann stattfinden:
  - In der Applikation
  - Im Vertex Shader
  - Gar nicht
  - …



5.3

Beispiel: Translation einer Kreisgeometrie

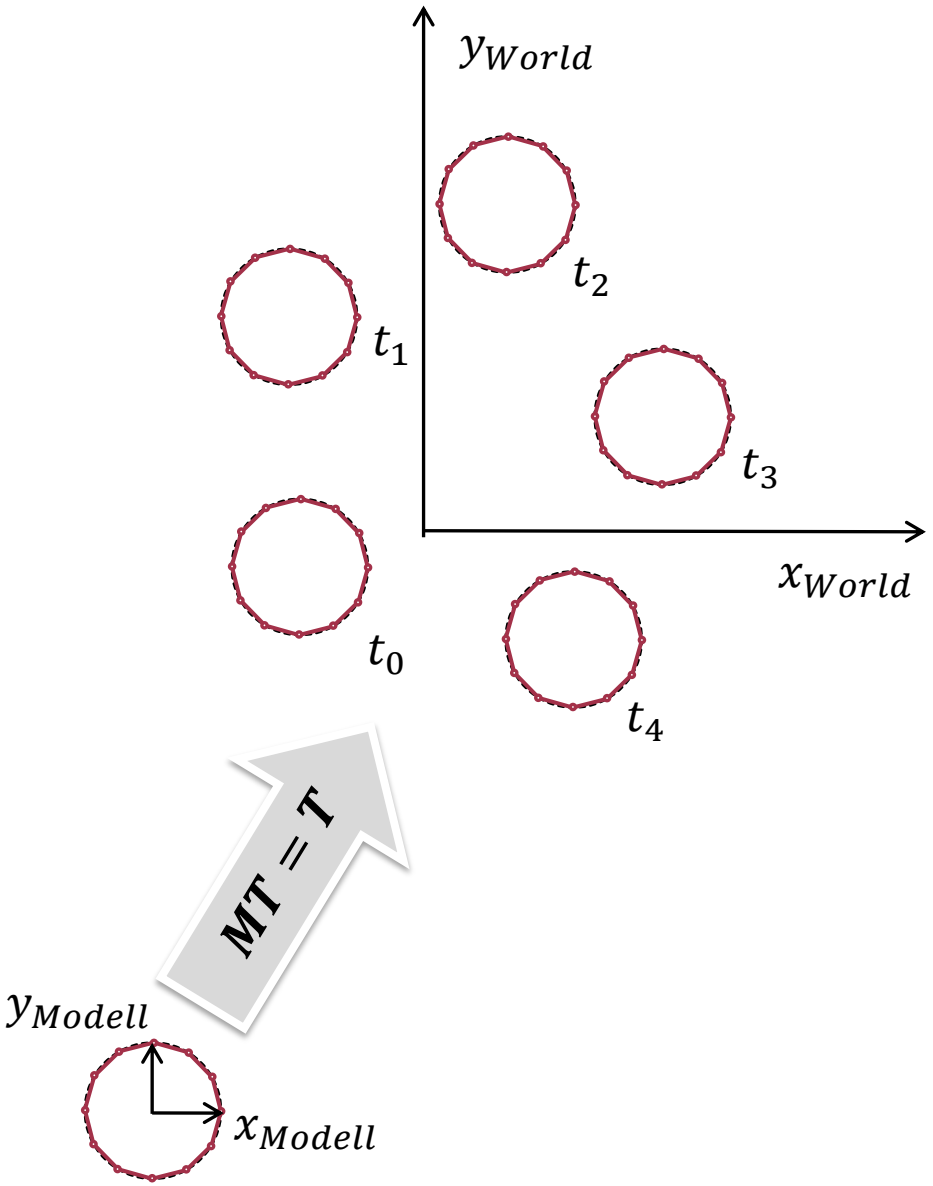
Beispielmodell eines Kreises



- Gegeben: Modell, hier: Kreis in xy-Ebene
- Genähert durch 12 Vertices
- Jeder Vertex besteht aus:
- Position "posMC" $(x_i, y_i, const, 1)$ in Modellkoordinaten

$$posMC_0 = \begin{pmatrix} 0{,}5 \\ 1 \\ const \\ 1 \end{pmatrix} \qquad posMC_3 = \begin{pmatrix} 1 \\ 0{,}5 \\ const \\ 1 \end{pmatrix}$$

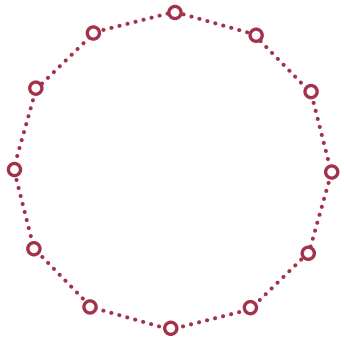
Modelling Transformation: Translation



- Ausgangsmodell wird durch Szene bewegt
- Modeling Transformation hier ausschließlich: Translation in xy-Ebene
- Translationsmatrix zeitabhängig
- Ziel: Ausführung im VS

Vertexdatenrepräsentation

- Pointer “vertices_in“ für Vertexdaten
- GPU: Hohe Datenbandbreite & Latenz
- Ziel: Gleichzeitig benötigte Daten konsekutiv speichern
- Enthält der Reihe nach:
  - Alle Komponenten…
  - Aller Attribute m
  - Aller Vertices n



Komponenten Attribut 0 des Vertex 0

Komponenten Attribut 1 des Vertex 0 …

$$vertices\_in = \{ p0_x, p0_y, p0_z, 1, \quad N/A \quad \texttt{// Daten von Vertex 0}$$
$$p1_x, p1_y, p1_z, 1, \quad \texttt{// Daten von Vertex 1}$$
$$p2_x, p2_y, p2_z, 1, \quad \texttt{// Daten von Vertex 2}$$
$$\ldots$$
$$p11_x, p11_y, p11_z, 1\}; \quad \texttt{// Daten von Vertex n-1}$$

5.4

Vertex Shader für Modeling Transformation

Unser Vorgehen

- Gegeben:
  - Position der Vertices der Vertices in Modeling Coordinates “`posMC`“
  - Attribut(e) sind hinterlegt globaler Datenstruktur “vertices_in“
  - Translationsmatrix: `trans`
- Gesucht:
  - Position der Vertices in World Coordinates “`posWC`“
  - Also ∀ Vertices berechnen: `posWC = trans * posMC;`
- Vorgehen:
  - Zunächst Pseudo VS, wie man es selbst implementieren könnte
  - Dann Übergang zu echtem Vertex Shader

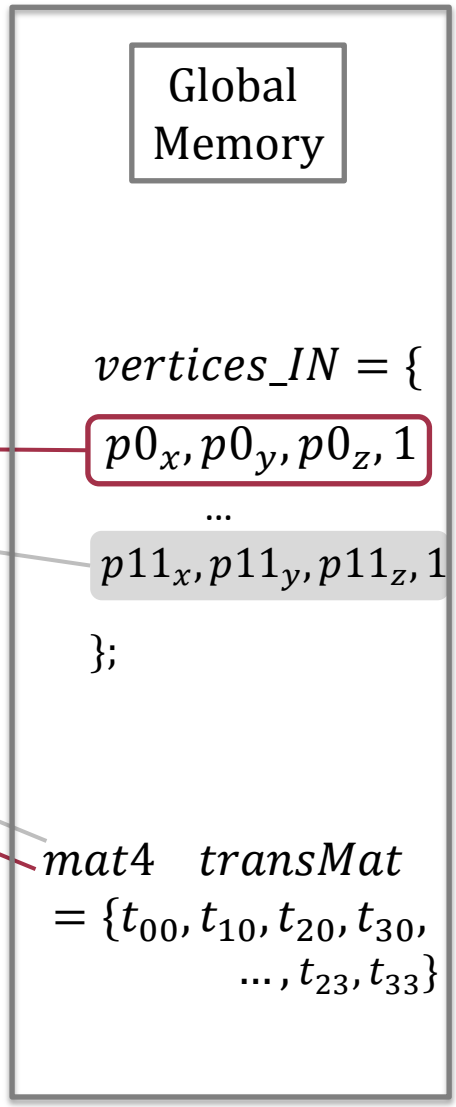
Pseudo Vertex Shader I

- Aufgabe des Vertex Shaders hier
  - Alle Vertices…
  - … transformieren
- Allgemein
  1. Lies Daten aller Vertices
  2. Verarbeite diese (optional)
  3. Reiche, eventuell verarbeitete, Daten weiter

```
const int vertexCnt = 12; // Unser Kreis, der hat 12 Ecken

for(int vertex = 0; vertex < vertexCnt; vertex++){

    <Read_Vertex_Data(vertex)>    // Folie 18

    <Do_Calculations(vertex)>     // Folie 19

    <Write_Data(vertex)>          // Folie 20

}
```

Pseudo Vertex Shader II

```
for(int vertex = 0; vertex < vertexCnt; vertex++){
    // Lokale Variablen
    vec4 posMC;
    vec4 posWC;
    mat4 trans;

    <Read_Vertex_Data(vertex)>
       posMC = readPosMC(vertex);
       trans = readTrans();

    <Do_Calculations(vertex)>
    <Write_Data(vertex)>
}
```



vertex = 0

vertex = 11

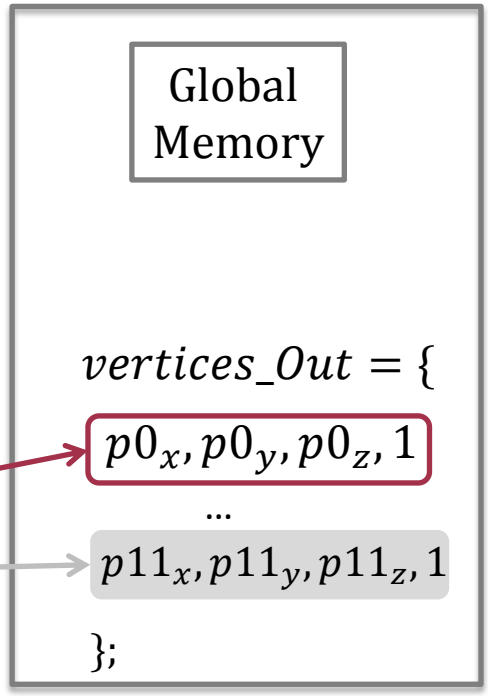
Pseudo Vertex Shader III

```
for(int vertex = 0; vertex < vertexCnt; vertex++){
    // Lokale Variablen
    vec4 posMC;
    vec4 posWC;
    mat4 trans;

    <Read_Vertex_Data(vertex)> // Done
    <Do_Calculations(vertex)>
       posWC = trans * posMC;
    <Write_Data(vertex)>
}
```

- Iteration `vertex = 0`:
  - Berechnet Produkt aus Matrix "`trans`" und `posMC` des 0-ten Vertex
- ...
- Iteration `vertex = 11`:
  - Berechnet Produkt aus Matrix "`trans`" und `posMC` des 11-ten Vertex

Pseudo Vertex Shader IV

```
for(int vertex = 0; vertex < vertexCnt; vertex++){
    // Lokale Variablen
    vec4 posMC;
    vec4 posWC;
    mat4 trans;

    <Read_Vertex_Data(vertex)> // Done
    <Do_Calculations(vertex)> // Done
    <Write_Data(vertex)>
       writePosWC(vertex, posWC);

}
```



vertex = 0 ← vertex = 11 ←

Pseudo Vertex Shader V

```
for all(int v in{0, ,vertexCnt-1} in parallel do{
// for(int v = 0; v < vertexCnt; v++){
// for(int v = vertexCnt-1; v >= 0; v--){
    // Lokale Variablen, haben eigenen Speicher
    // per Iteration
    vec4 posMC;
    vec4 posWC;
    mat4 trans;
    <Read_Vertex_Data(v)>
       posMC = readPosMC(v);
       trans = readTrans();
    <Do_Calculations(v)>
       posWC = trans * posMC;
    <Write_Data(v)>
       writePosWC(v, posWC);
}
```

- Verschiedene R/W Kategorien der Daten
- "posMC": **in**
  - Read-Only
  - Nimmt je Vertex anderen Wert an…
  - …und wird nur für diesen verwendet
- "trans": **uniform**
  - Read-Only
  - Wert ∀ Vertices gleich
- "posWC": **out**
  - Write-Only
  - Nimmt je Vertex anderen Wert an…
  - …und wird nur für diesen verwendet

- Zugriffe auf globalen Speicher der Schleifeniterationen unabhängig
- ⟹ Alle Iterationen der Schleife sind unabhängig voneinander
- ⟹ Können parallel ausgeführt werden

Pseudo Vertex Shader VI

```
// Unser Pseudo VS zur Translation
// beliebiger Geometrie
for all(int v in{0, ,vertexCnt-1}
                    in parallel do{
    vec4 posMC;
    vec4 posWC;
    mat4 trans;
    <Read_Vertex_Data(v)>
       posMC = readPosMC(v);
       trans = readTrans();

    <Do_Calculations(v)>
       posWC = trans * posMC;

    <Write_Data(v)>
       writePosWC(v, posWC);
}
```

```
// Entsprechender echter
// Vertex Shader
#version 330 core

in vec4 posMC;
uniform mat4 trans;
out vec4 posWC;

void main(){

   posWC = trans * posMC;

}
```

Shader, hier: Vertex Shader

```
// Für alle Vertices der Geometrie geschieht parallel:
//   1. Erzeuge eine Instanz dieses (Vertex) Shaders
//   2. Lade dessen Daten in die in Variablen aus globalem Speicher
//   3. Führe darauf das Programm (Methode main) aus
//   4. Schreibe Ergebnisse in out deklarierte variablen
//   5. Schreibe out Variablen in globalen Speicher
// Programmierer hat lediglich Einfluss auf Schritte 3 und 4.
#version 330 core

in vec4 posMC;       // Per-Instanz (hier: Vertex) Daten aus vorherigem
                     // Pipeline Schritt

uniform mat4 trans; // Global lesbare Daten. Für alle Instanzen gleich

out vec4 posWC;      // Per-Vertex Daten zur weiteren Verarbeitung im
                     // nächsten Pipeline Schritt
void main(){
  posWC = trans * posMC; // Berechnungen, basierend auf in und uniform
                // deklarierten Daten. Schreibe diese in out-Variablen.
}
```

Build-In Variablen

- OpenGL kennt Bedeutung der Build-In Variablen …
- … und benötigt/liefert sie für/durch nicht programmierbare Abschnitte der Pipeline
- Implizit deklariert

```
#version 330 core

in vec4 posMC;
uniform mat4 trans;
// out vec4 glPosition; So sähe die Deklaration von glPosition aus.
// Entsprach im letzten Beispiel out vec4 posWC;

void main(){
  // Schreibe build-in out Variable glPosition.
  // Position der Vertices wird z.B. zum Backface-Culling und
  // Rastern benötigt.
  // Daher muss OpenGL die Bedeutung dieser Variable kennen.
  glPosition = trans * posMC;
}
```