# Rendering Pipeline

## Medientechnik 5

# Rendering Pipeline - Übersicht

Vertex Spezification

Vertex Shader

Tessellation

Geometry Shader

Vertex Post-Processing

Primitive Assembly

Rasterization

Fragment Shader

Per-Sample Operations

Programmierbare
Shader Phase (Stage)

# Rendering Pipeline - Übersicht

Vertex Spezification

Vertex Shader

Tessellation

Geometry Shader

Vertex Post-Processing

Primitive Assembly

Rasterization

Fragment Shader

Per-Sample Operations

# Vertex Spezification

- ist der Prozess, bei dem die Applikation eine geordnete Liste von Vertices bereitstellt, um sie an die Pipeline zu senden.
- Vertex-Attributes definieren die Eigenschaften der Primitiven.
- Die Primitiven sind grundlegende geometrische Formen, bestehend aus Dreiecken, Linien und Punkte.
  - Übersetzungen: Vertex (Vertices) = Eckpunkt€, Rand = Boundary, Primitive = Grundform (z.B. Dreieck)
- Die Interpretation der Liste von Vertices als Primitiven wird in einer späteren Phase der Pipeline behandelt.
- Die Vertex Spezification beschäftigt sich mit folgenden Objekten:

  - **Vertex Array Objects** (VAOs)
    Speichert die Definition aus welchen Daten (Vertex-Attributes) sich jeder Vertex zusammensetzt (Koordinaten, Farbe, Transparenz), also das Format.
    Außerdem beinhaltet es die Buffer Objects, mit den eigentlichen Vertex-Daten (VBOs).

  - **Vertex Buffer Objects** (VBOs)
    Eigenschaften jedes einzelnen Vertex (X, Y, [Z, r, g, b, a])

Vertex Spezification

Vertex Shader

Tessellation

Geometry Shader

Vertex Post-Processing

Primitive Assembly

Rasterization

Fragment Shader

Per-Sample Operations

htl Hollabrunn

# Vertex Spezification

- **Vertex Array Objects** (VAOs)
  Speichert die Definition aus welchen Daten (Vertex-Attributes) sich jeder Vertex zusammensetzt (Koordinaten, Farbe, Transparenz), also das Format.
  Außerdem beinhaltet es die Buffer Objects, mit den eigentlichen Vertex-Daten (VBOs).

- **Vertex Buffer Objects** (VBOs)
  Eigenschaften jedes einzelnen Vertex (X, Y, [Z, r, g, b, a])

```
struct Vertex {
    Glfloat position[3];
    Glfloat normal[3];
    Glubyte color[4];
};

Vertex vertices[VERTEX_COUNT];
```

```
struct StructOfArrays {
    Glfloat positions[VERTEX_COUNT * 3];
    Glfloat normals[VERTEX_COUNT * 3];
    Glubyte colors[VERTEX_COUNT * 4];
};

StructOfArrays structOfArrays;
```

# Vertex Spezification

- Nach der genauen Beschreibung (Spezifikation) der Vertex-Daten, werden diese als Primitive mittels Drawing-Calls gerendert
  (to render – ausführen, übergeben)

  → Vertex Rendering

Vertex Spezification

Vertex Shader

Tessellation

Geometry Shader

Vertex Post-Processing

Primitive Assembly

Rasterization

Fragment Shader

Per-Sample Operations

# Vertex Shader

Shader? 🤔

- kleines benutzerdefiniertes Programm das auf der Grafikeinheit ausgeführt wird
- hoher Grad an Parallelität
- verantwortlich für die Berechnung
  - der einzelnen Vertices – Vertex Shader
    $$f(x, y, z) \rightarrow (x, y, z)$$
  - der einzelnen Primitiven – Geometry Shader (optional)
    $$f(primitive) \rightarrow (primitive[])$$
  - der Farbe jedes einzelnen Pixels – Fragment Shader
    $$f(x, y) \rightarrow (r, g, b, a)$$

Vertex Spezification

Vertex Shader

Tessellation

Geometry Shader

Vertex Post-Processing

Primitive Assembly

Rasterization

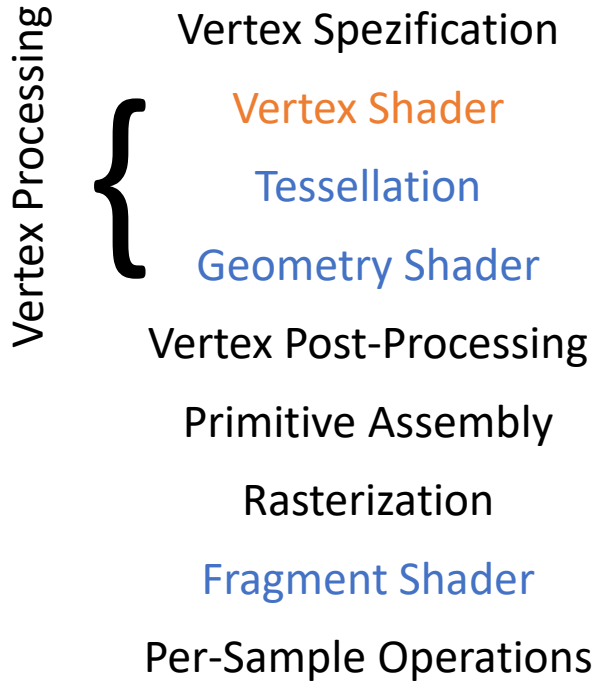
Fragment Shader

Per-Sample Operations

htl Hollabrunn

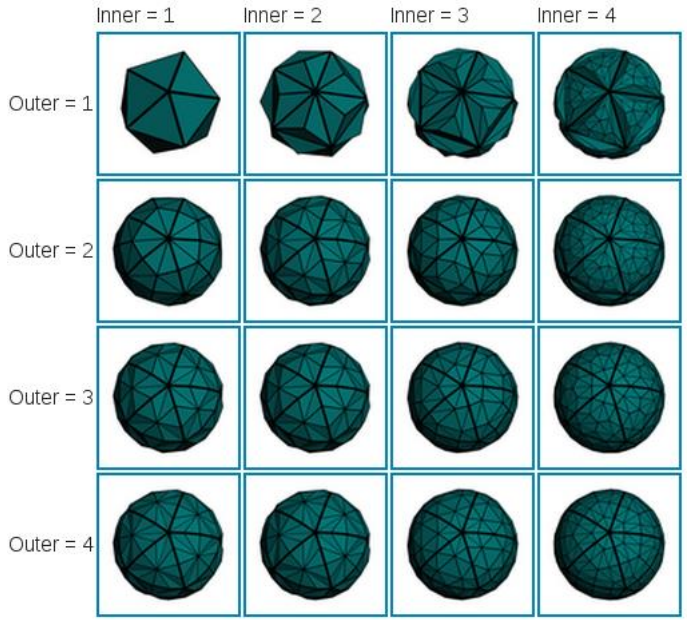
# Vertex Shader

- Erster Schritt des **Vertex Processing**, also der Verarbeitung der definierten Vertices aus dem Vertex Rendering, ist der Vertex Shader.
- Alle Phasen des Vertex Processing sind programmierbare Operationen, sodass eine individuelle Verarbeitung der Vertices möglich wird.
- Der Vertex Shader ist eine Funktion mit einem Vertex als Eingangsparameter und einem Vertex als Rückgabetyp.
  **Sprachen:** OpenGL Shading Language **GLSL** bei OpenGL,
  High-Level Shader Language **HLSL** bei DirectX von Microsoft
- Vertex Shader sind nicht optional,
  einfachster Vertex Shader: Input = Output
- Eine Limitierung beim Vertex Processing ist, dass jeder Input Vertex auf einen Output Vertex gemappt werden muss. Es können weder neue Vertices generiert werden noch bestehende verworfen werden.



# Tessellation (Mosaik)

- Ein Tessellation-Shader besteht aus zwei getrennten Shadern und einem Fixed-Function Tessellator.
- Er zerlegt Flächen (definiert als sogenannte Patches) in kleinere Flächen.




Vertex Spezification

Vertex Shader

Tessellation

Geometry Shader

Vertex Post-Processing

Primitive Assembly

Rasterization

Fragment Shader

Per-Sample Operations

# Geometry Shader

- Ein Geometry Shader wird pro Primitve aufgerufen und erhält dieses als Input.
- Das Ergebnis eines Geometry Shaders können 0 oder mehr Primitives beinhalten.
- Dadurch kann neue Geometrie erstellt werden und beispielsweise Tessellation stattfinden.
- Vertex-Attribute können ebenfalls verändert werden z.B. Vertex Positionen interpolieren.
- GS können auch Geometry-Typen verändern
  z.B.
  Punkt → Dreieck
  Linien → Punkte
  usw.

Vertex Spezification

Vertex Shader

Tessellation

Geometry Shader

Vertex Post-Processing

Primitive Assembly

Rasterization

Fragment Shader

Per-Sample Operations

htl Hollabrunn

# Vertex Post-Processing – Primitive Assembly

- Nach der Shader-basierenden Vertex-Verarbeitung folgen einige statische Funktionen:

1. Transform Feedback (Vertex Post-Processing)
   Vertex-Daten von GPU zurück an CPU liefern (optional, Rückgabe-Buffer vorbereiten notwendig)
2. Primitive Assembly
   Sammeln der Output-Vertex-Daten aus den vorigen Phasen und Erstellen einer Sequenz von Primitiven.
3. Clipping
   Clipping bedeutet, dass Primitiven, die am Rand des sichtbaren Bereichs liegen bzw. den Rand überschreiten, aufgesplittet werden. Nicht sichtbare Primitiven werden verworfen.
4. Face Culling
   (cull – Auslese, Wegschneiden)
   Nur Primitiven die in Richtung des Window-Space zeigen werden gerendert – jene deren Normale um mehr als 90° abweicht werden verworfen.

Vertex Spezification

Vertex Shader

Tessellation

Geometry Shader

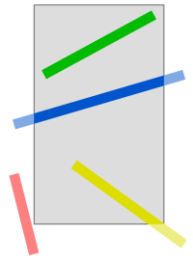
Vertex Post-Processing

Primitive Assembly

Rasterization

Fragment Shader

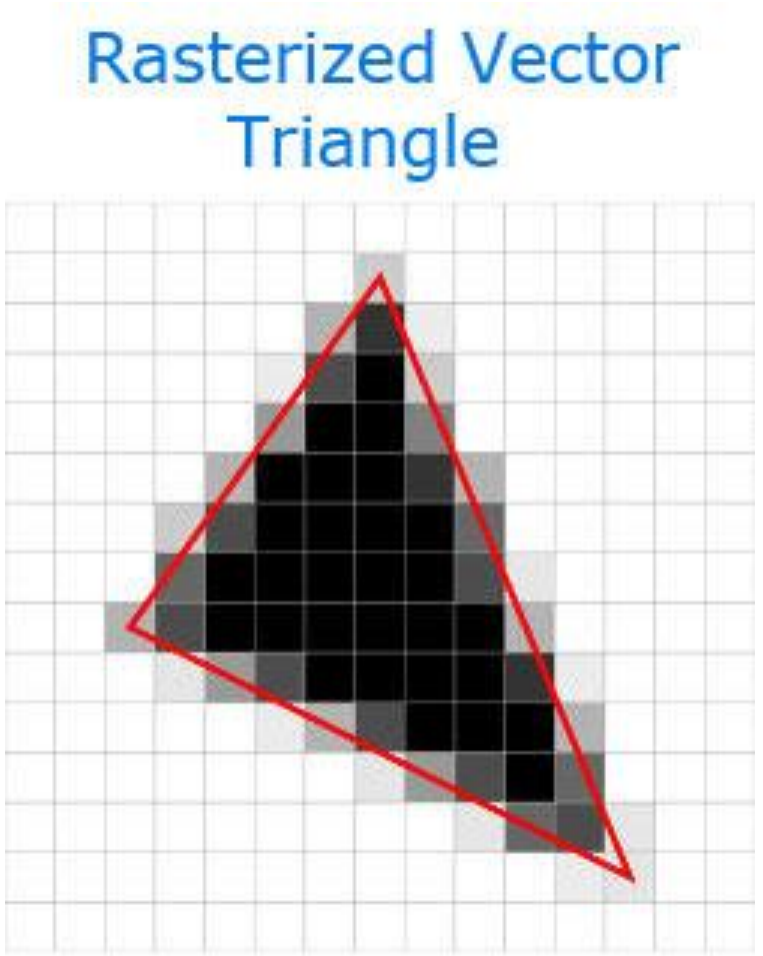
Per-Sample Operations



htl Hollabrunn

# Rasterization

- Alle Primitiven die bis in diese Phase gelangen werden gerastert.
- Rastern bedeutet ein Bild durch ein enges Netz von sich kreuzenden Linien in viele einzelne Punkte zerlegen.
- Kommt auch bei der Konvertierung von Vektorgrafiken in Pixelgrafiken zum Einsatz.
- Der Abstufung der Grundfarbe ergibt sich dadurch, wie viel Prozent des gesamten Pixels vom Primitive belegt ist.




*https://www.computerhope.com/jargon/r/rasterize.htm*

Vertex Spezification

Vertex Shader

Tessellation

Geometry Shader

Vertex Post-Processing

Primitive Assembly

Rasterization

Fragment Shader

Per-Sample Operations

# Rasterization

- Multisampling verwendet weitere Iterationen bei denen "Sub-Pixel" nach Berücksichtigung der Z-Werte berechnet werden, sofern die Differenz der Z-Werte einen Schwellwert überschreitet. Dies tritt nur an Kanten von Primitiven auf.
- Z-Werte befinden sich im Tiefenbuffer (Depth-Buffer). Dieser wird bei Überlappung von Primitiven verwendet, um zu entscheiden, welche Farbe sichtbar ist.
- Ergebnis der Rasterisierung ist eine Sequenz von Fragmenten (Pixel). Dabei wird zwischen den relevanten Vertex-Daten Interpolation eingesetzt.
- Das Fragment beinhaltet Daten, die Vertex Shader bzw. Geometry Shader berechnet wurden, die Position im Screen Space (x/y).

Vertex Spezification

Vertex Shader

Tessellation

Geometry Shader

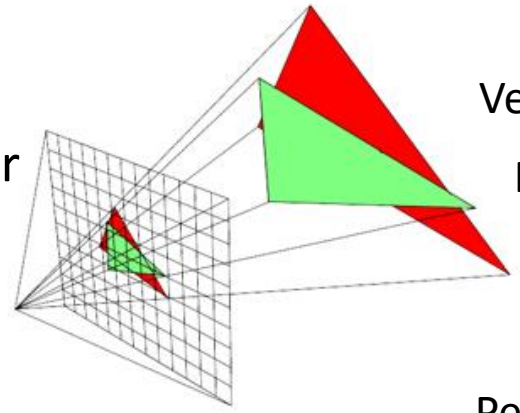
Vertex Post-Processing

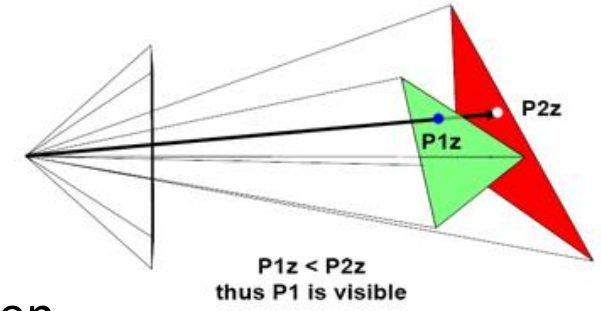
Primitive Assembly

Rasterization

Fragment Shader

Per-Sample Operations






https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation/overview-rasterization-algorithm.html

# Fragment Shader

- Alle Fragmente (Pixel) die in der Sequenz der Rasterisierung enthalten sind, werden vom Fragment Shader verarbeitet.
- Das Ergebnis des Fragment Shaders besteht aus Farbwert für den Color Buffer, Depth Value und Stencil Value.
- Depth Values → Depth Buffer
- Stencil Values (stencil = Schablone) → Stencil Buffer. Sie können nicht vom Fragment Shader verändert werden.

Vertex Spezification

Vertex Shader

Tessellation

Geometry Shader

Vertex Post-Processing

Primitive Assembly

Rasterization

Fragment Shader

Per-Sample Operations

htl Hollabrunn

# Per-Sample Operations

- Abschließend finden erneut einige statische Operationen statt (Culling Tests).
- Diese Tests müssen aktiviert werden.

1. Pixel-Ownership-Test:
   Prüft, ob das Pixel OpenGL gehört. Liegt beispielsweise ein anderes Fenster über der 3D-Szene wird das geprüfte Pixel nicht angezeigt.
2. Scissor-Test:
   Prüft, ob das Pixel innerhalb eines definierten Rechtecks im Screen Space liegt.
3. Stencil-Test:
   Prüft, ob das zugehörige Bit im Stencil Buffer gesetzt ist.
4. Depth-Test:
   Prüft, ob ein definierter Vergleich des Z-Werts erfolgreich ist.
   Initial: GL_LESS → Fragmente näherer Primitiven werden in den Framebuffer geschrieben.


Vertex Spezification
Vertex Shader
Tessellation
Geometry Shader
Vertex Post-Processing
Primitive Assembly
Rasterization
Fragment Shader
Per-Sample Operations

# Quellen

- https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview
- https://registry.khronos.org/OpenGL-Refpages/gl4/
- https://learnopengl.com/