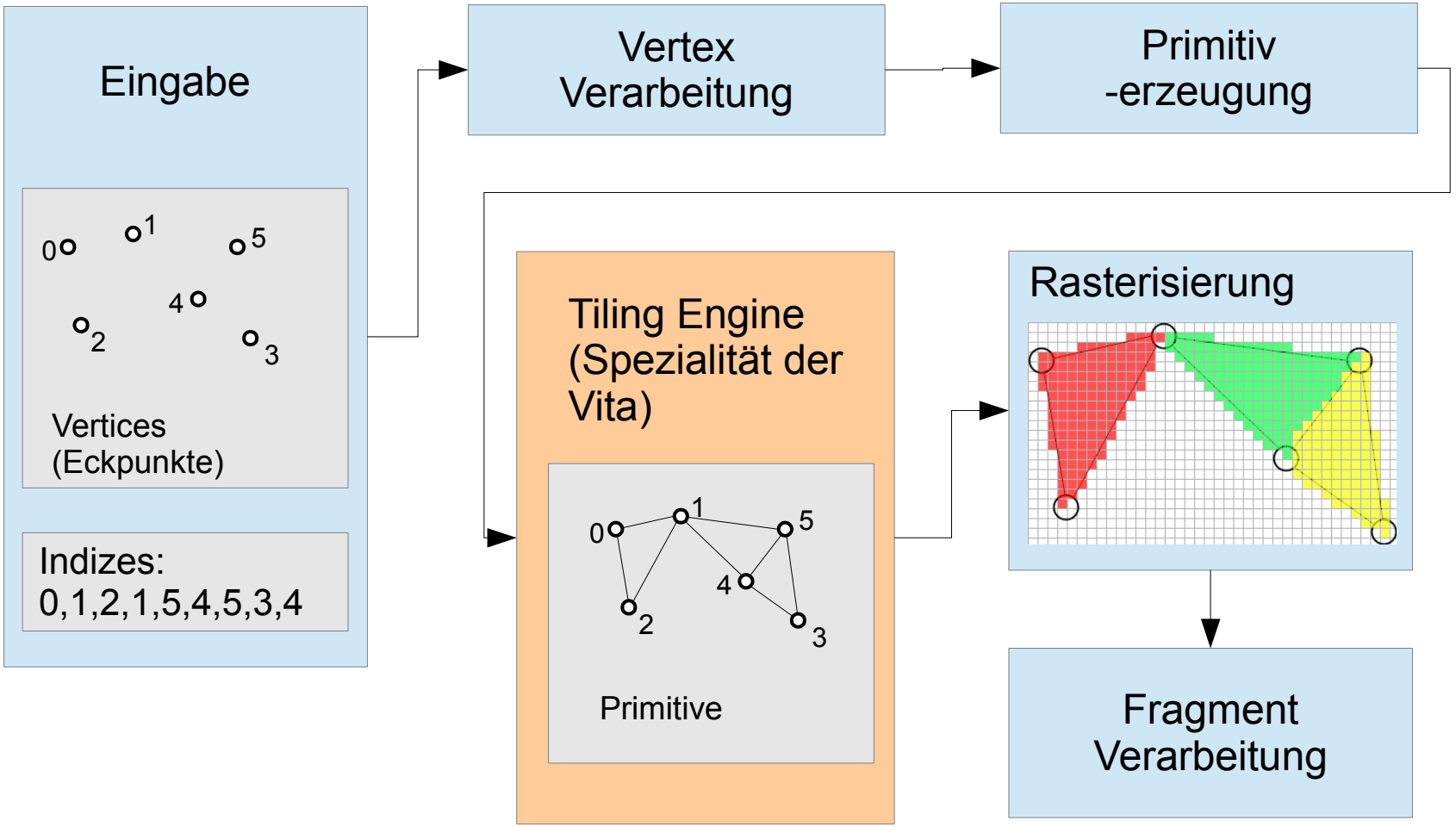
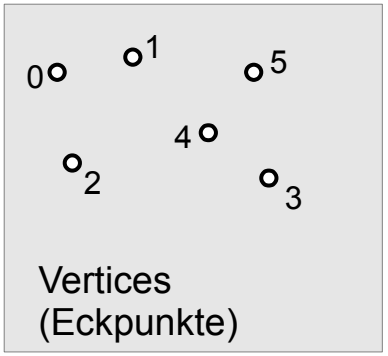
Tutorium
Computergrafik PS Vita

Inhalt

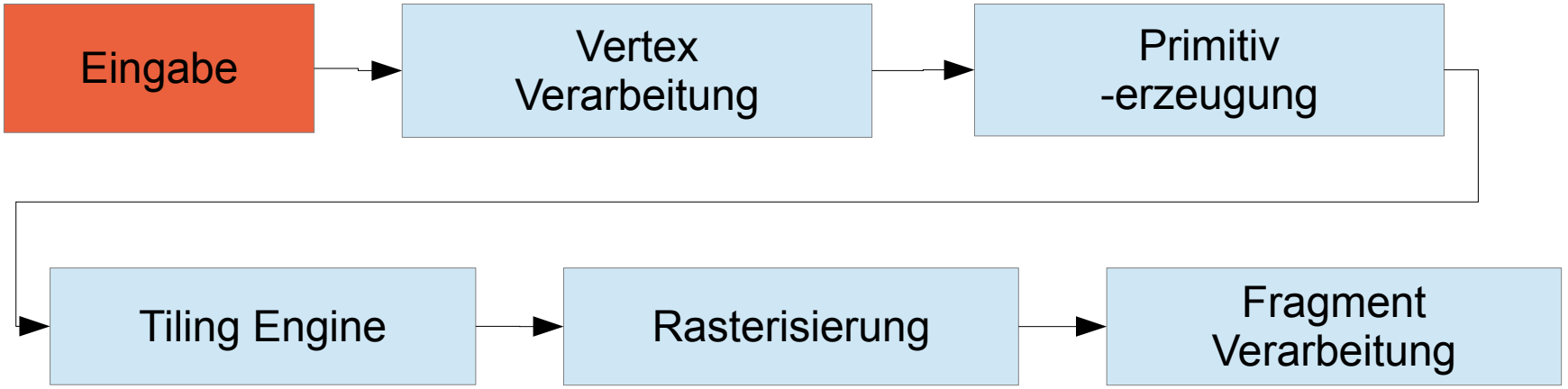

Eingabe
0
1
2
3
4
5
Vertices
(Eckpunkte)
Indizes:
0,1,2,1,5,4,5,3,4
Vertex
Verarbeitung
Primitiv
-erzeugung
Tiling Engine
(Spezialität der
Vita)
Primitive
Rasterisierung
Fragment
Verarbeitung

Eingabe

- GPU ist auf zeichnen von Dreiecken ausgelegt.
- Werden erzeugt aus:
  - Vertices: Eckpunkte der Geometrie
  - Indizes: bestimmen welche Eckpunkte einzelne Dreiecke bilden

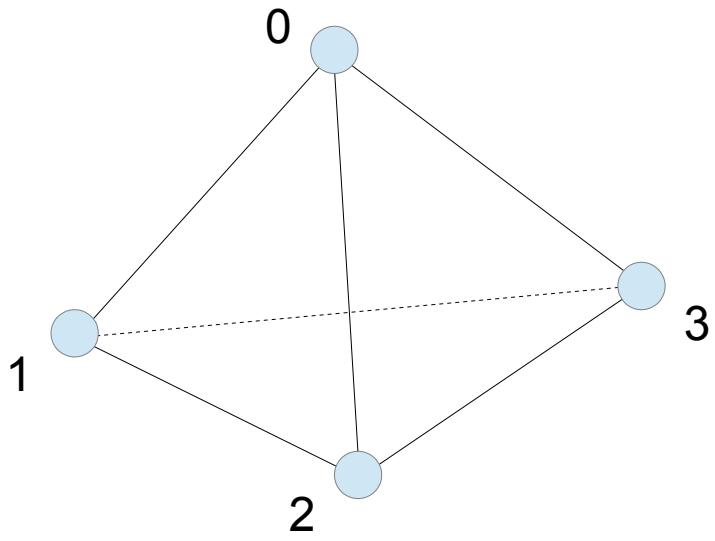


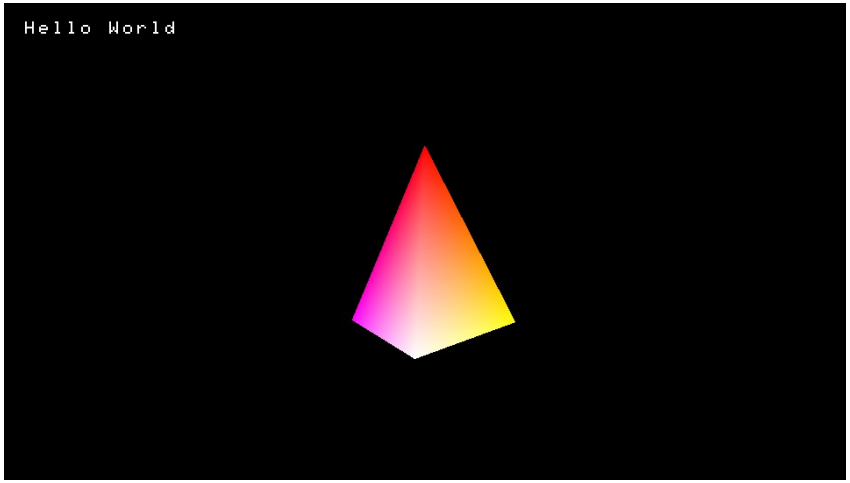
Indizes:
0,1,2,1,5,4,5,3,4



Übung 1

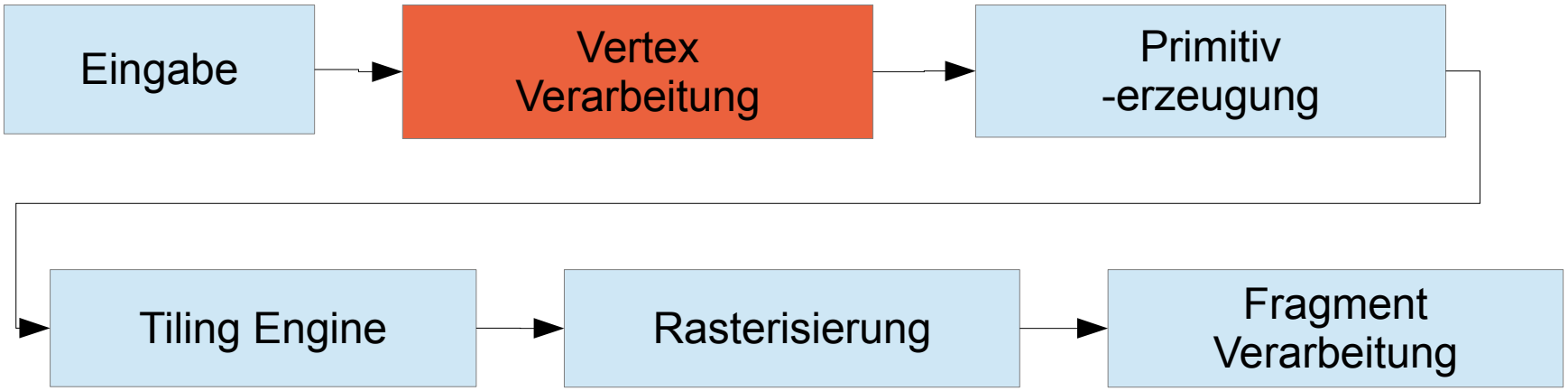
- TetraederBase.zip enthält ein Skelett das die Vertices für einen Tetraeder enthält.
- Die Vertices entsprechen ungefähr der Darstellung rechts.
- Indizes fehlen.
- Ergänze die fehlenden Indizes um die einzelnen Seiten des Tetraeders aufzubauen.





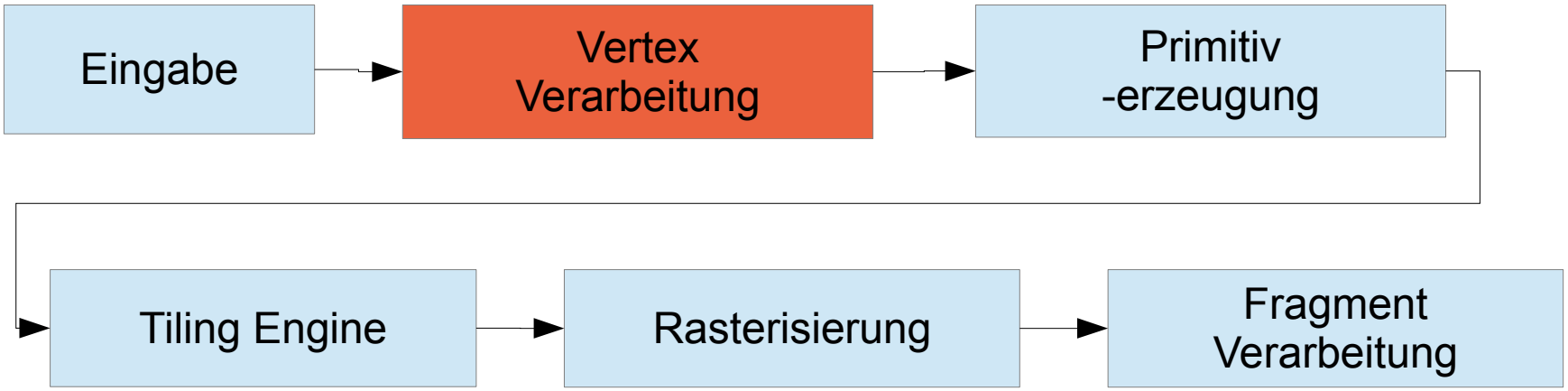
Vertex Verarbeitung

- Erste programmierbare Pipeline-Stufe.
- Vertex kann beliebige Daten enthalten.
- Vertex kann beliebig interpretiert werden.
- Ausgabe mindestens Clip-Space Position.
  - Würfel dessen xy-Kanten von -1 bis 1 reichen, z von 0 bis 1
- Zusätzliche Ausgaben können selbst spezifiziert werden



Uniforms

- Zusätzlich zu Vertices und Indizes ist es möglich für alle Vertices konstante Daten anzugeben.
- z.B.:
  - Matrizen
  - Vektoren
- Daten können im Shader beliebig interpretiert werden

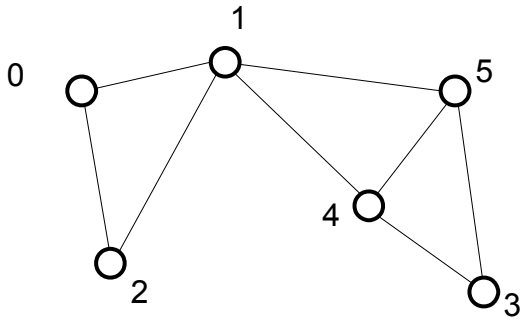


Aufgabe 2

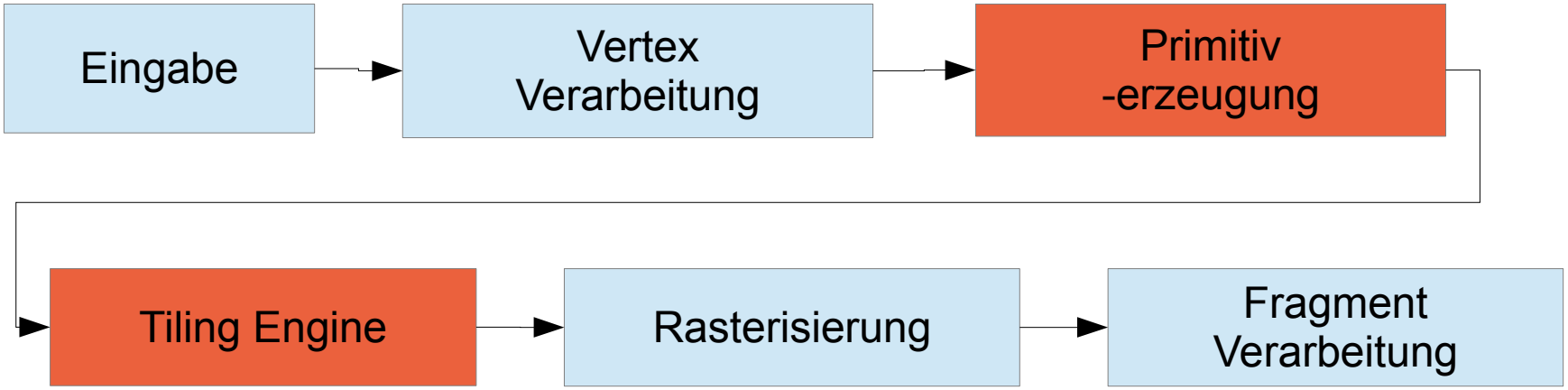
- TriangleRotationBase.zip enthält ein Programm das ein Dreieck zeichnet.
- Der Vertex-Shader basic_v.cg enthält einen Uniform-Parameter der die vergangene Zeit enthält.
- Nutze diesen Parameter und die Vertex-Position um das Dreieck im Vertex-Shader zu rotieren.
- Erweitere anschließend die Vertex-Definition um einen zusätzlichen Parameter mit dem festgelegt werden soll ob ein Vertex im Shader gedreht werden soll oder nicht.
- Nutze anschließend im Vertex-Shader diesen Parameter um nur einen einzigen Vertex zu rotieren.

Primitiv-erzeugung / Tiling Engine

- Erzeugung geometrischer Primitive.
  - Dreiecke (List, Strips, Fans)
  - Linien
  - Punkte

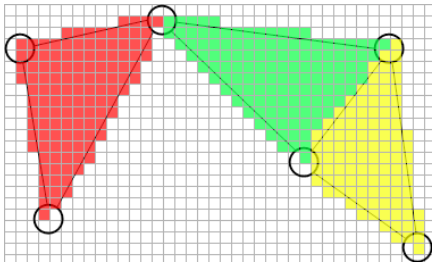


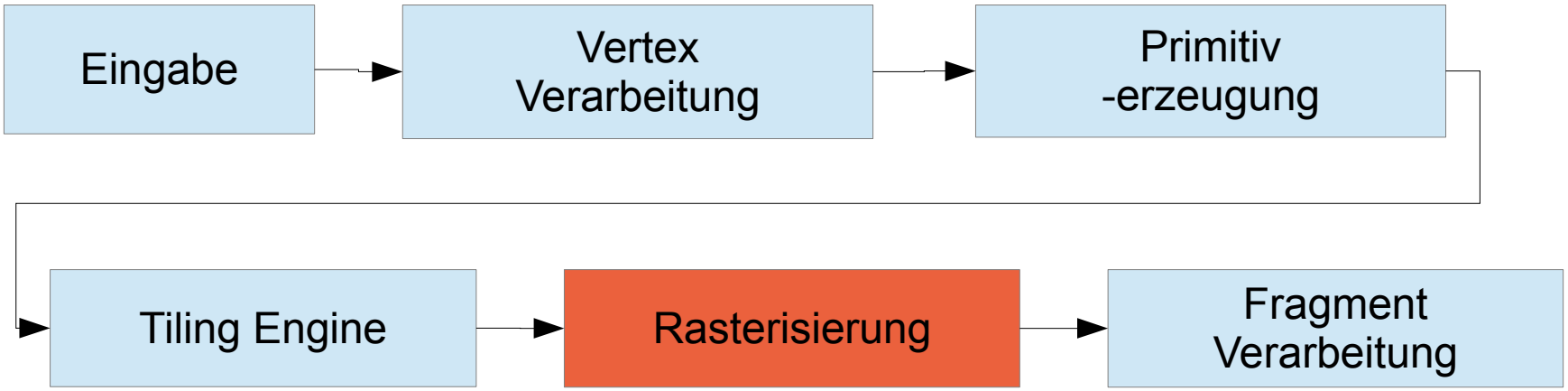
- Werden in Tile-System gespeichert.
  - Bei Vertex-Verarbeitung entstandene Ausgaben werden auch hier gespeichert.



Rasterisierung

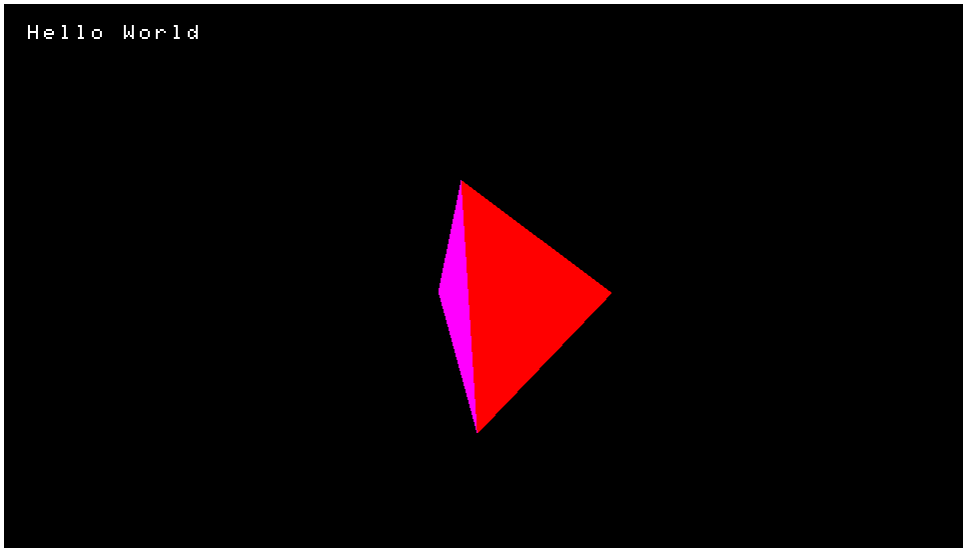
- Darstellung an diesem Punkt in Form von Primitiven.
- Display kann mit dieser Darstellung nichts anfangen.
- Zerlegung der Primitive in Fragmente.
- Interpolation von Vertex-Processing Ausgaben. (z.B. Texturkoordinaten)





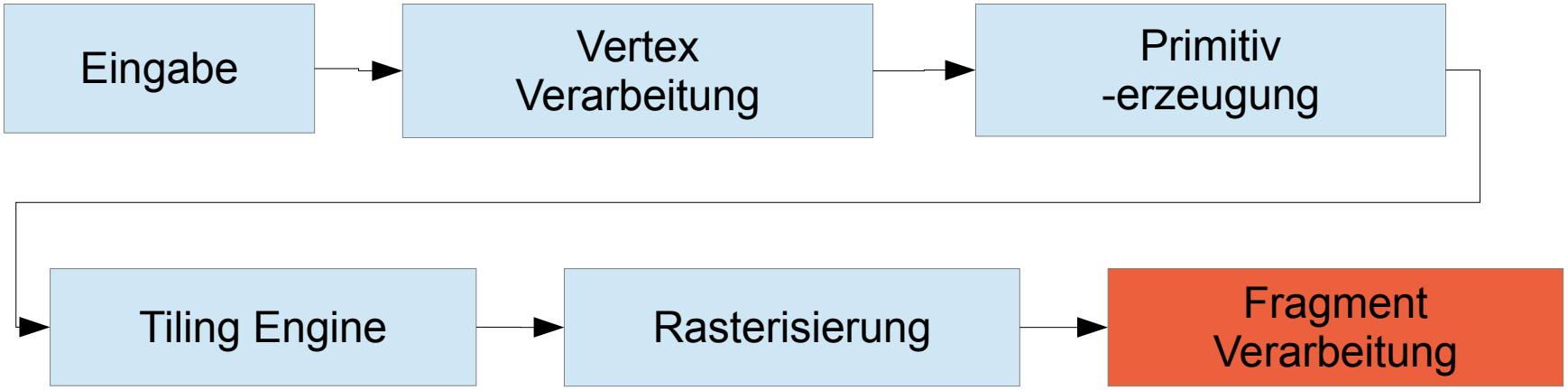
Aufgabe 3

- Nehmen sie den fertigen Tetraeder aus Aufgabe 1
- Bei diesem soll nun jede Fläche einfarbig werden.
- Wie lässt sich die Interpolation der Werte verhindern?
- Wie muss der Vertex-Buffer hierzu geändert werden?



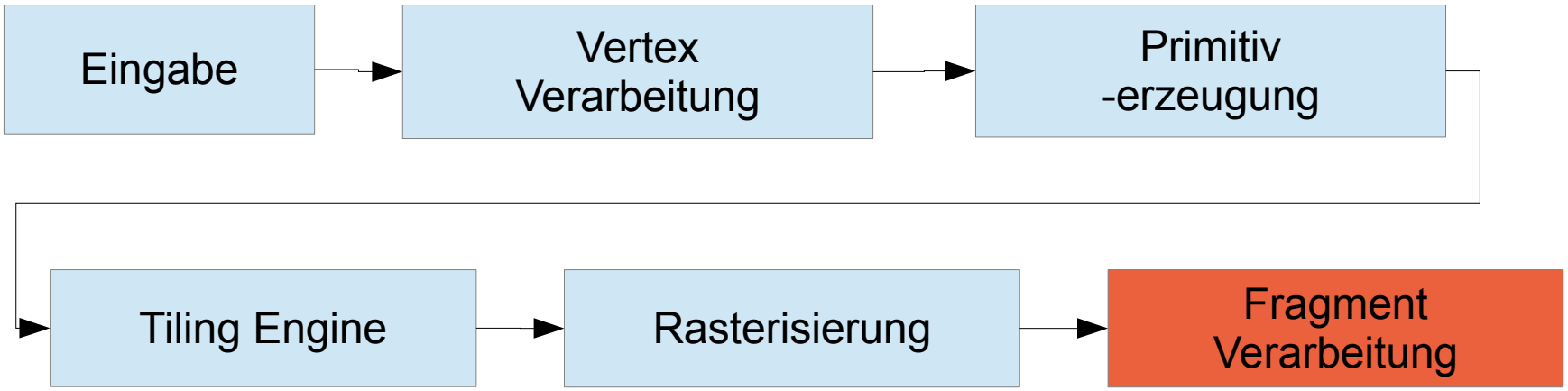
Fragment Verarbeitung

- Weitere frei programmierbare Stufe
- Besitzt lediglich eine einzige Ausgabe: Farbe
- Benötigt theoretisch keinerlei Eingaben. Jedoch in Praxis häufig:
  - Normale (Beleuchtungsberechnung)
  - Texturkoordinate



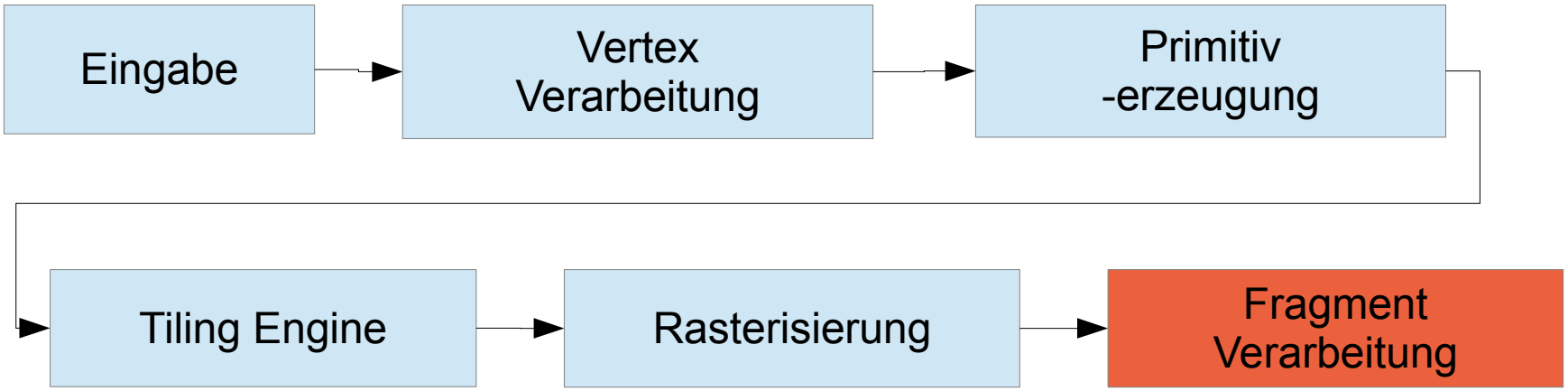
Texturen

- Rastergrafiken können in Fragment-Shader gelesen werden.
- Hierzu muss jedes Fragment jedoch wissen aus welchem Bereich der Rastergrafik gelesen werden soll.
- Definiere für jeden Vertex Position in Textur.
  - wird während Rasterisierung interpoliert.
- Texturkoordinaten können beliebig interpretiert werden → siehe nächste Übung.



Texturen

- Rastergrafiken können in Fragment-Shader gelesen werden.
- Hierzu muss jedes Fragment jedoch wissen aus welchem Bereich der Rastergrafik gelesen werden soll.
- Definiere für jeden Vertex Position in Textur.
  - wird während Rasterisierung interpoliert.
- Texturkoordinaten können beliebig interpretiert werden → siehe nächste Übung.



Aufgabe 4

- Öffne die Solution in FragmentShadingBase.zip
- Momentan wird lediglich ein rotes Viereck gezeichnet.
- Erweitere die Vertex-Struktur um Texturkoordinaten.
- Gebe zum Testen der Koordinaten, diese als Farbe im FragmentShader aus.
- Verwende diese Koordinaten um in jeder Ecke des Vierecks eine andere Farbe zu zeichnen. Der Übergang zwischen den Farben soll scharf sein, Farben sollen nicht interpoliert dargestellt werden.