Vertex (mehrz. *vertices*)

Beschreibt einen Punkt im grafischen 3D-Raum. Ein Vertex besteht meistens aus der Position im 3D-Raum (*Vector3f*), der Farbe (*Color4f*), und einer Texturkoordinate (damit OpenGL weiß wie eine Texture auf ein Objekt gelegt werden soll)(*Vector2f*)

Texturkoordinaten = UV = TexCoord = TexKoordinaten

Vector3f

Ein Vektor mit 3 Komponenten, x, y & z. Wird u.a. dafür verwendet Positionen oder Skalierungen im 3-dimensionalen Raum zubeschreiben.

Hier entspricht dann z.B. die x Komponente die Distanz auf der X-Achse zum Ursprung, bzw. die Skalierung auf der x-Achse.

Vector2f

Ein Vektor mit 2 Komponenten, x & y. Wird u.a. dafür verwendet Positionen oder Skalierungen im 2-dimensionalen Raum zubeschreiben.

Color4f

Ein Vektor mit 4 Komponenten, r, g, b, & a. Wird dafür verwendet eine Farbe zubeschreiben. r entspricht dem Rot-Anteil, g dem Grün-Anteil, b dem Blau-Anteil, und a der Sichtbarkeit. Der Wertebereich für r,g,b & a liegen zwischen 0.0f und 1.0f.

a = 1.0f entspricht voller Sichtbarkeit, 0.0f kompletter Transparenz, und 0.5f halber Sichtbarkeit/Transparenz.

Quaternion4f

Wird dazu benutzt um Rotationen zubeschreiben, ohne die Rotation als Verkettung von Rotation über die (Euler-)Winkel (Pitch, Roll und Yaw) zubetrachten. Grund hierfür ist, dass die Betrachtung von Rotationen als Verkettung von Rotationen um Eulerwinkel zu einen *Gimbal-Lock* führen kann.

Quaternionen können miteinander multipliziert werden um Rotationen zu kombinieren (Quaternionen-Multiplikationen sind wie Matrix-Multiplikationen NICHT kommutativ)

(Wie genau eine Rotation in Quaternionen gespeichert/repräsentiert wird weiß ich nicht mehr, ich werde ggf. Quellen zu dem Thema noch ergänzen.)

Matrix4f

Eine 4 x 4 Matrix. Wird im Allgemeinen dazu verwendet um eine Transformation (*siehe Transform*) im 3-dimensionalen Raum zubeschreiben. Matrizen werden insofern gerne verwendet da man konkrete Punkte im 3D-Raum die als Vector3f dargestellt sind einfach mit einer Matrix multiplizieren kann und damit auch schon die komplette beschriebene Transformation auf den Punkt angewendet wurde.

(Achtung: Matrix ist 4x4, die Vektoren jedoch nur 1x3, das heißt wir müssen vor jeder Matrix-mit-Vektor Multiplikation den Vektor in einen Vektor 1x4 umwandeln. Dies geht ganz einfach, man übernimmt die 3 Komponenten wie sie sind und „hängt" eine vierte Komponente mit dem Wert 1.0f an. Dies beeinflusst nicht das Resultat. Pseudo-Code Bsp: matrix * vec4(vector.x, vector.y, vector.z, 1.0f))

Transform

Beschreibt eine Transformation im 3-dimensionalen Raum. Teil einer Transformation ist die Beschreibung der Position (Translation), der Skalierung (Scaling) und der Rotation (Rotation).

Die Translation wird repräsentiert durch einen *Vector3f.*

Das Scaling wird repräsentiert durch einen *Vector3f.*

Die Rotation wird repräsentiert durch einen *Quaternion4f.*

Tileset

Ein Tileset ist ein Bild dass mehrere Unterbilder besitzt. Diese Unterbilder werden von eine TileMap als Tiles dargestellt. Als Beispiel das Tileset von Minecraft (http://s44.photobucket.com/user/Dante80/media/Minecraft/terrain-6.png.html).

Die Unterbilder eines Tileset's werden typischerweise durchnummeriert (beginnend von oben -links nach oben rechts, und von oben nach unten 0,1,2,...,*n*.

TileMap

Eine 2D-Spielwelt wird oft in rechteckige Bereiche unterteilt. Am besten stellt man sich das wie eine Mauer vor (daher auch der Name *tile, englisch* für *Ziegel*). Eine TileMap ist eine Ansammlung von Tiles die dann zusammen eine zusammenhängende Welt suggerieren soll.

Eine TileMap hat ein 2D-Array indem an jedem Platz eine Zahl zwischen 0 und *n* steht, die einem Unterbild im dazugehörigen Tileset entspricht. Die TileMap zeichnet nun für jede Zahl im Array an dem entsprechenden Platz das dazugehörige Unterbild des Tilesets, so entsteht ein zusammenhängendes, allerdings sehr flexibel modifizierbares Gesamtbild.

Modelspace

Viewspace

Projectionspace

VertexArrayObject / VAO

Wird in OpenGL benutzt um weitere Render-Objekte zubündeln. Enthält sonst keine weiteren Daten. Enthält immer ein VBO. Kann enthalten IndexBuffer oder InstanceBuffer.

BufferObject

Ist ein Puffer im Grafikkartenspeicher.

VertexBufferObject / VBO

Enthält ein Buffer mit Vertices

Ist ein Puffer im Grafikkartenspeicher der dafür benutzt wird um Vertices zuspeichern.

IndexBufferObject / IBO

Enthält ein Buffer mit Integers

Da öfters Vertices mehrfach genutzt werden, will man den Speicher sparen und Vertices nicht mehr als einmal in einem VBO haben. Ein IBO enthält Indices die von 0 aufsteigend auf die Vertices im VBO abbilden. So kann man dann dem Renderer eine Liste von Vertices, und eine Reihenfolge von Indizes geben und der Renderer rendert diese (und man spart sich einiges an Speicherplatz und Datenkommunikation zwischen CPU und GPU)

InstanceBufferObject / IBO (ist inkonsitent, werde das im Code mal umbenennen damit es keine Verwechslung mit IndexBufferObject gibt)

Enthält ein Buffer mit Matrix4f's

Es kommt hin und wieder vor dass man ein und dasselbe Objekt, identisch an vielen Plätzen darstellen will (Achtung, die Objekte müssen exakt identisch sein, nicht ein Attribut darf abweichen, ansonsten ist InstancedRendering nicht möglich, zumindest nicht mit unserem Code). Dies erreicht man indem man die Vertexdaten für ein Objekt in einen VBO speichert und dann separat für jede Instanz eine Transformationsmatrix übergibt. Der Renderer rendert dann das übergebene Objekt das im VBO liegt einmal kombiniert mit jeder Transformationsmatrix die im InstanceBufferObject liegt.

Shader

Auch Shader-Program genannt, ist eine Kompilation von mehreren Shadermodulen. Diese legen fest wie Vertices durch die Grafikkarte visualisiert werden. Ein Shadermodul wird in einer C ähnlichen Sprache geschrieben „GLSL" (GL Shading Language).

Vertexshader

Wird als erstes Shadermodul ausgeführt. Es legt die Position fest an dem der Vertex gerendert wird.

Die Position an der der Vertex gerendert werden soll muss der vordefinierten Variable *gl_Position* zugewiesen werden.

```
#version 330                // Welche GLSL Version benutzt wird

uniform mat4 uni_wvp;       // Die Model/View/Projection-Matrix

// layout: Position des Attributpointers; in: Eingehende Variable mit Wert; vec2/vec3/vec4/mat4… GLSL-Typen
layout(location = 0) in vec3 vs_position;// Die Attributposition in dem VBO mit der Position
layout(location = 1) in vec4 vs_color;    // Die Attributposition in dem VBO mit der Farbe
layout(location = 2) in vec2 vs_texcoord;// Die Attributposition in dem VBO mit der Tex-Koordianten

out vec4 fs_color;          // Hier stellen wir 2 out-Variablen bereit deren Werte an den Fragment-
out vec2 fs_texcoord;       // Shader weiter gegeben werden


void main() {
    gl_Position = vec4(vs_position, 1.0) * uni_wvp; // Position des Vertex * MVP
    fs_color = vs_color;          // Farbe & Tex-Koordinaten an den Fragmentshader weiter
    fs_texcoord = vs_texcoord;  // reichen
}
```

Fragmentshader

Wird als zweites Shadermodul ausgeführt. Es legt die Farbe fest die der Vertex hat.

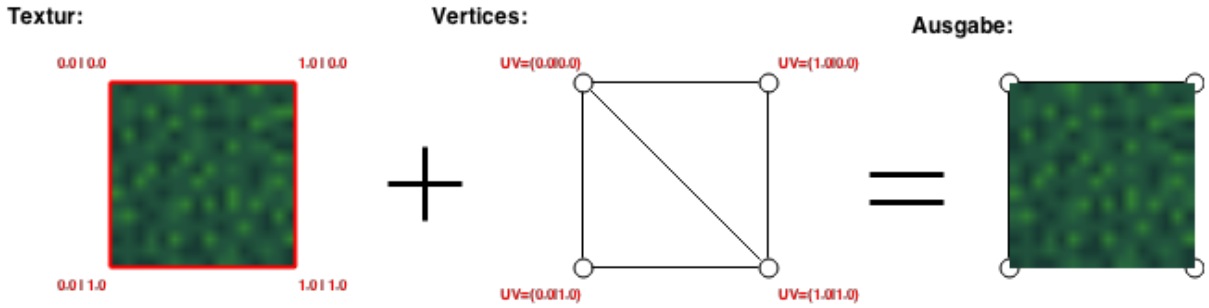
```
#version 330 // Welche GLSL Version benutzt wird

uniform sampler2D uni_texture0; // Texture Slot 0, an den eine Texture gebunden werden kann

in vec4 fs_color;
in vec2 fs_texcoord;

out vec4 out_color;

void main() {
    out_color =  texture2D(uni_texture0,fs_texcoord) * (fs_color + vec4(0.5,0.5,0.5,0));
}
```

Material

Besteht aus Shader und Texture.

Texture

Ein Bild das mittels eines Shaders „auf Vertices gelegt" wird. Die UV Werte eines Vertex geben an welche Texturkoordinate an den Vertex gemapped werden.



Double-/TripleBuffering / BackBuffer / FrontBuffer

DoubleBuffering: Es gibt 2 Buffer, die je einen Farbwert pro Pixel aufnehmen.

FrontBuffer: Ist der Buffer der auf dem Bildschirm dargestellt wird, in dem jedoch momentan nicht gezeichnet wird.

BackBuffer: Ist der Buffer der momentan nicht auf dem Bildschirm dargestellt wird, in den jedoch gezeichnet wird.

Buffer A, Buffer B:

1: Buffer A : FrontBuffer | Buffer B : BackBuffer

2: DrawCalls werden getätigt => malen auf den BackBuffer

3: SwapBuffers: BackBuffer wird zu FrontBuffer, FrontBuffer wird zu BackBuffer

4: Frontbuffer wird auf den Bildschirm gezeichnet

5: Wiederhohle Schritt 2

DrawCall

Sobald Vertices an OpenGL übergeben wurden, Texturen an TexturUnits gebunden wurden, ein Shader als aktiver Shader gesetzt wurde und dann eine Funktion mit der Signatur *glDraw…* aufgerufen wird, dann werden die Vertices durch den Shader *gejagt* und in den BackBuffer gemalt.

Batch

Abstrahiert die OpenGL Renderschicht (VAO, VBO, IBO, InstancedBuffer). Es werden Vertices und Material übergeben und daraus werden *DrawCalls* erzeugt.

StreamingBatch

- Beliebig viele, verschiedene Vertices

- Ein Material für alle Vertices

InstancingBatch

- Vertices die exakt ein Objekt darstellen

- Ein Material

- *Transform* ‘s die eine Instanz des Objekts repräsentieren

GameObject

Actor

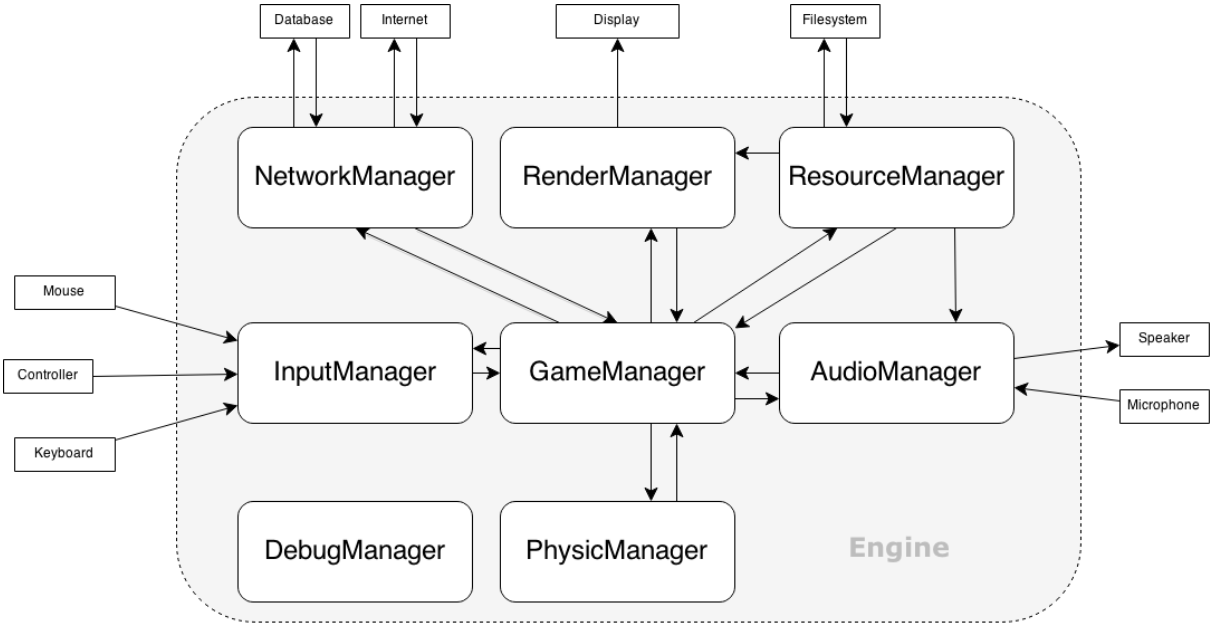
Abstrahiert die Batches. Hier kommen zum Beispiel Actors wie *TileMap, Image2D, Text* oder so.

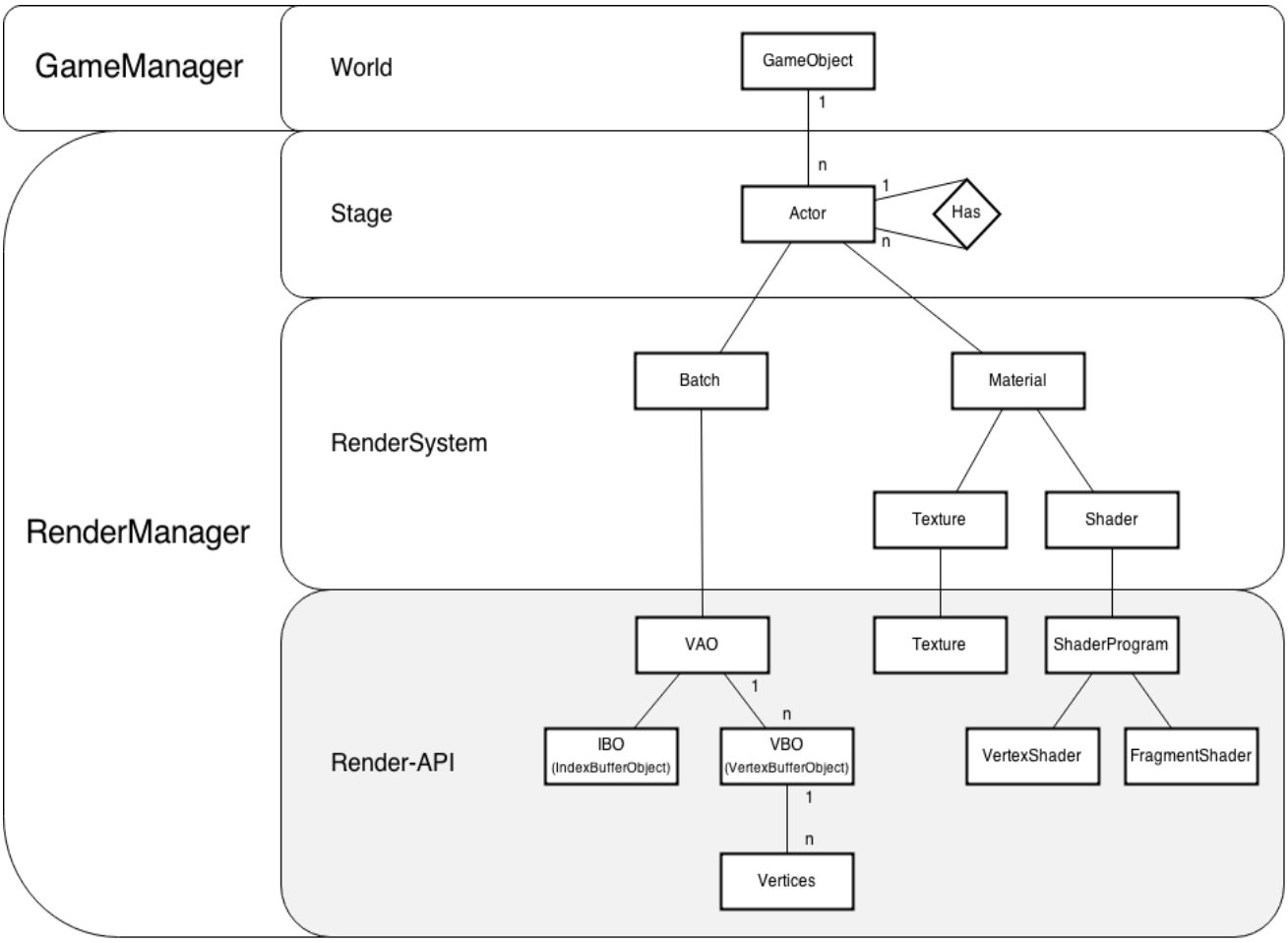
Module

LogicModule

RenderModule

Manager


Database
Internet
Display
Filesystem
NetworkManager
RenderManager
ResourceManager
Mouse
Controller
Keyboard
InputManager
GameManager
AudioManager
Speaker
Microphone
DebugManager
PhysicManager
Engine


GameManager
World
GameObject
1
n
Stage
Actor
1
Has
n
Batch
Material
RenderSystem
Texture
Shader
RenderManager
VAO
Texture
ShaderProgram
1
n
Render-API
IBO
(IndexBufferObject)
VBO
(VertexBufferObject)
VertexShader
FragmentShader
1
n
Vertices