

# Computergraphik II

## Übung 01

### OpenGL Teil 1: Vertex-Attribute und Buffer-Objekte


第一次实验课程的目标是了解用于硬件加速渲染的现代OpenGL的一些基本原理。在这里，我们建立在CG1的基础上 - 您应该原则上知道渲染管道的结构和操作。

Das Ziel der ersten Übungseinheiten ist das Kennenlernen einiger Grundprinzipien von modernem OpenGL zum hardwarebeschleunigten Rendern. Wir bauen hier auf die Kenntnisse aus der CG1 auf - Ihnen sollte Aufbau und Funktionsweise einer *Render-Pipeline* prinzipiell bekannt sein.

在本课中，我们将按照讲座中的示例，使用顶点缓冲对象（VBO）和顶点阵列对象（VAO）来扩展场景并相应地向GPU提供数据。我们提供了一个软件框架，用于创建OpenGL 4.3 Core Profile渲染上下文。

In dieser Einheit werden wir das Beispiel aus der Vorlesung nachvollziehen und Vertex Buffer Objects (VBO) und Vertex Array Objects (VAO) verwenden um die Szene zu erweitern, und der GPU entsprechend Daten zur Verfügung zu stellen. Wir stellen ein Software-Framework zur Verfügung, welches einen OpenGL 4.3 Core Profile Rendering-Kontext erstellt.<sup>1</sup>

Tastaturbelegung:

-  : Programm beenden

## 1. Vertex Buffer Objects und Vertex Array Objects

该框架仍然对应于讲座的状态 - 呈现单色三角形，其将下两个角连接到窗的上边缘的中间。为了练习生成缓冲区和配置顶点数组对象，应首先为三角形赋予颜色属性。在cg2application.cpp中的void CG2App::init\_gl\_state()方法中，您已经找到了一个数组GLubyte colors[9] = {...}，它包含三角形每个顶点的相应颜色。

Das Framework entspricht noch dem Stand aus der Vorlesung - es wird ein einfarbiges Dreieck gerendert, welches die Unteren beiden Ecken mit der Mitte des oberen Fensterrandes verbindet. Um das Erzeugen der Buffer und die Konfiguration der Vertex Array Objects zu üben soll das Dreieck zunächst mit einem Farbattribut versehen werden. In der Methode `void CG2App::init_gl_state()` in `cg2application.cpp` finden Sie bereits ein Array `GLubyte colors[9] = {...}`, welches für jeden Vertex des Dreiecks die entsprechende Farbe beinhaltet. Die einzelnen Komponenten der Farben (Rot, Grün und Blau) werden durch `GLubyte` (`unsigned char` also Werte im Bereich `[0, 255]`) repräsentiert, wobei 0 der minimalen und 255 der maximalen Intensität entspricht.

颜色（红色，绿色和蓝色）的各个组成部分由GLubyte（范围[0; 255]中的无符号字符值）表示，其中0对应于最小强度，255对应于最大强度。

- Erzeugen Sie das Buffer Object `color_bn` welches die Farbwerte speichert und kopieren Sie die Farben in diesen Buffer. (`glGenBuffers`, `glBindBuffer`, `glBufferData`).

创建缓冲区对象`color_bn`，它存储颜色值并将颜色复制到此缓冲区中。（`GlGenBuffers`，`glBindBuffer`，`glBufferData`）。

- Erweitern sie das Vertex Array Objekt um einen weiteren Vertex Attribute Pointer, sodass neben der Position (Attribut 0) aus dem Buffer `bn` auch die Farbe (Attribut 1) aus den buffer `color_bn` gelesen wird. Achten Sie darauf, dass die Farbwerte als Integer angegeben sind, und entsprechend normiert werden müssen. (`glVertexAttribPointer`)

使用另一个顶点属性指针扩展顶点数组对象，以便除了缓冲区`bn`的位置（属性0）之外，还从缓冲区`color_bn`读取颜色（属性1）。确保将颜色值指定为整数，并且必须相应地进行标准化。（`GlVertexAttribPointer`）

- Passen Sie den Vertex Shader (`data/shaders/example.vs.glsl`) so an, dass das Farbattribut `clr` als Varying `vclr` weitergegeben wird.

调整顶点着色器（`data / shaders / example.vs.glsl`）以将颜色属性`clr`作为Varying `vclr`传递。

<sup>1</sup> Dies setzt geeignete GPUs und Treiber voraus: Nvidia seit “Fermi”-Generation (2011, ab Geforce 420), AMD Radeon seit “Terascale 2”-Architektur (2009, ab Radeon HD 5000), Intel ab “Haswell”-Prozessorgeneration (2013, ab Intel HD 4200). Unter Linux stellt Nvidia eigene proprietäre Treiber bereit, AMD und Intel-GPUs werden über das OpenSource-Projekt [Mesa](#) mit geeigneten Treibern versorgt, ab Mesa 17.x sollte damit auch geeigneter OpenGL 4.3 Support für die meisten neueren GPUs verfügbar sein. MacOS unterstützt OpenGL nur bis 4.1 und wird daher von unserem Framework gar nicht unterstützt!

**void glGenBuffers(GLsizei n, GLuint\* buffers)** [↗](#)

Erzeugt  $n$  neue Buffernamen und schreibt diese an die Adresse buffers.  
生成N个新缓冲区名称并将其写入地址缓冲区。

**void glBindBuffer(GLenum target, GLuint buffer)** [↗](#)

Bindet den Buffer mit dem Namen buffer an das angegebene target. Andere GL-Funktionen beziehen sich auf Buffer, die an bestimmte Targets gebunden sind. Folgende Targets sind in dieser Übung relevant: 将缓冲区与名称缓冲区绑定到指定的目标。 其他GL函数指的是绑定到特定目标的缓冲区。 以下目标与本练习相关：

**GL\_ARRAY\_BUFFER:** Der Buffer in dem sich Vertex Attribute befinden. 顶点属性所在的缓冲区。  
**GL\_ELEMENT\_ARRAY\_BUFFER:** Der Buffer in dem sich die Vertex Indices befinden. 顶点索引所在的缓冲区。

**void glBufferData(GLenum target, GLsizeiptr size, const GLvoid\* data, GLenum usage)** [↗](#)

Alloziert size Bytes Speicher an dem, an target gebundenen Buffer, und kopiert size Bytes aus data in diesen Buffer. Ist data == nullptr wird nur Speicher Angelegt aber kein Speicher kopiert. Der usage-Hint beschreibt, wie der Buffer verwendet werden wird, speziell wie oft die Daten verändert werden. Im Rahmen der CG2 kommen drei verschiedene Anwendungsfälle in Frage:

**GL\_STREAM\_DRAW:** Daten werden einmal gesetzt und dann seltene Male im Rendering verwendet. 数据设置一次，然后很少用于渲染。  
**GL\_STATIC\_DRAW:** Daten werden einmal gesetzt und dann häufig im Rendering verwendet. 数据设置一次，然后在渲染中频繁使用。  
**GL\_DYNAMIC\_DRAW:** Daten werden ständig gesetzt und ständig im Rendering verwendet. 数据不断被设置并不断用于渲染。

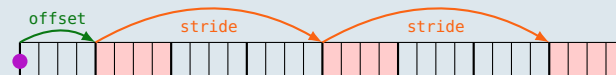
**void glVertexAttribPointer(GLuint index, GLint elements, GLenum type, GLboolean normalized, GLsizei stride, const GLvoid\* offset)** [↗](#)

Beschreibt das Format und die Lage des Vertex Attributs index im aktuell an **GL\_ARRAY\_BUFFER** gebundenen Buffer. Das Format wird durch die ersten drei Parameter beschrieben:

**elements:** Die Anzahl der Elemente des Attributs im Buffer. Sind im Buffer bspw. Die Positionswerte  $x, y, z$  gespeichert sind das drei Elemente des Vektors.  
**type:** Der Typ in dem die Werte vorliegen. Im Shader werden die Attribute in float-Vektoren konvertiert, wir können die Werte aber auch in anderen Formaten kodieren.  
**normalized:** Bezieht sich auf die Interpretation von Integertypen. Wird bspw. ein Attribut mit dem Typ **GL\_UNSIGNED\_BYTE** kodiert können die Werte ([0-255]) entweder direkt als float interpretiert werden als Werte im Bereich [0.0f, 255.0f], oder normiert [0.0f, 1.0f] mit 255 Abstufungen. Integerdatentypen mit  $b$ -Bit werden also nach folgendem Schema interpretiert:

normalized	unsigned i	signed i
<b>GL_TRUE</b>	$f := \frac{i}{2^b - 1}$	$f := \max\{\frac{i}{2^{b-1} - 1}, -1\}$
<b>GL_FALSE</b>	$f := i$	$f := i$

Die Lage der Werte im Buffer wird mit den verbleibenden Parametern angegeben: stride beschreibt dabei den Abstand zwischen zwei aufeinander folgenden Instanzen des Attributs in Bytes und offset beschreibt die Position der ersten Instanz, relativ zum Bufferanfang.



**void glEnableVertexAttribArray(GLuint index)** [↗](#)

Aktiviert das Vertex Array für das Attribut index. Dieser Zustand wird im Vertex Array Object gespeichert. 为index属性启用顶点数组。 此状态存储在顶点数组对象中。

## 2. Objekttransformation

在框架的当前状态中，三角形独立于窗口大小显示，从而显示下窗口角落中的两个角落和窗口上边缘的剩余角落。

Im aktuellen Zustand des Frameworks wird das Dreieck unabhängig von der Fenstergröße so dargestellt, dass zwei Ecken in den unteren Fensterecken und die verbleibende Ecke am oberen Fensterrand dargestellt wird.

分析着色器代码：如果矩阵MVP在每个帧中是常数，那么在哪个空间中给出三角形的坐标？MVP矩阵的价值是多少？

- (a) Analysieren sie den Shader-Code: In welchem Raum sind die Koordinaten des Dreiecks gegeben, wenn die Matrix MVP in jedem Frame konstant ist? Welchen Wert hat die Matrix MVP?

在cg2application.cpp中调整方法void CG2App::render\_one\_frame()，使三角形围绕y轴旋转。变量time\_stamp包含自程序启动以来经过的时间（以秒为单位），可以直接用于生成旋转。

- (b) Passen Sie die Methode `void CG2App::render_one_frame()` in `cg2application.cpp` so an, dass sich das Dreieck um die Y-Achse dreht. Die Variable `time_stamp` beinhaltet die vergangene Zeit seit Start des Programms in Sekunden und kann direkt zur Erzeugung der Rotation verwendet werden.

继续调整MVP矩阵，使摄像机朝向(3, 3, 3) T并朝向原点。为此，应使用透视投影（打开角度：60°）。

- (c) Passen Sie die Matrix MVP weiter an, sodass die Kamera in  $(3, 3, 3)^T$  steht und in Richtung des Ursprungs schaut. Dazu soll eine perspektivische Projektion verwendet werden (Öffnungswinkel: 60°).



对于向量和矩阵的表示，我们使用库glm。除此之外，它还提供了表示各种向量（`glm::vec2`, `glm::vec3`, `glm::vec4`）和矩阵（`glm::mat2`, `glm::mat3`, `glm::mat4`）的类型。相应的函数可用于生成转换矩阵：

Zur Repräsentation von Vektoren und Matrizen verwenden wir die Bibliothek *glm*. Diese liefert, unter anderem, Typen zur Repräsentation von diversen Vektoren (`glm::vec2`, `glm::vec3`, `glm::vec4`) und Matrizen (`glm::mat2`, `glm::mat3`, `glm::mat4`). Zu Erzeugung von Transformationsmatrizen stehen entsprechende Funktionen zur Verfügung:

```
glm::mat4 glm::rotate(float a_rad, glm::vec3 axis)
```

Erzeugt eine Rotationsmatrix, welche eine Rotation um `a_rad` um `axis` beschreibt.

```
glm::mat4 glm::translate(glm::vec3 offset)
```

Erzeugt eine Translationsmatrix mit einer Verschiebung um `offset`.

```
glm::mat4 glm::lookAt(glm::vec3 eye, glm::vec3 center, glm::vec3 up)
```

Erzeugt eine entsprechende Viewmatrix.

```
glm::mat4 glm::perspective(float fovy_rad, float aspect_ratio, float near, float far)
```

Erzeugt eine Projektionsmatrix passend zum angegebenen horizontalen Öffnungswinkel (`fov_y`), Seitenverhältnis (`aspect_ratio`), und den `near` und `far` Werten.

**ALLE WINKEL WERDEN IN RADIANT ANGEGBEN!**

```
float glm::radians(float deg)
```

Rechnet einen Winkel von Grad in Radiant(Bogenmaß) um.

## 3. Strukturen im Vertex Array

到目前为止，我们已经为每个属性使用了单独的数组，并将它们放在不同的缓冲区中。

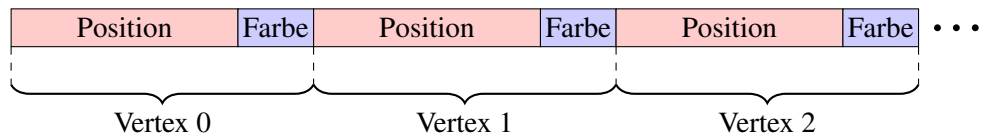
Bisher haben wir mit separaten Arrays für jedes Attribut gearbeitet, und diese in verschiedene Buffer abgelegt.

运行顶点着色器时，GPU无论如何都必须为所有已启用的属性数组加载数据。因此，实际上，如果顶点的所有属性数据一起存储在存储器中，则通常更有效。

Bei der Ausführung des Vertex-Shaders muss die GPU ohnehin die Daten für alle aktivierten Attribut-Arrays laden. Daher ist es in der Praxis meist effizienter, wenn alle Attributdaten eines Vertex zusammen im Speicher abgelegt werden.

- (a) Um die Attribute eines Vertex möglichst nah beieinander abzulegen sollen die Attributarrays *verschachtelt* werden. Durch Wahl geeigneter offsets und stride-Parameter lassen sich so auch mehrere Attributarrays (*interleaved*) im Speicher ablegen, also z.B.:

为了使顶点的属性尽可能彼此接近，应该嵌套属性数组。通过选择合适的偏移和跨参数，可以将多个属性数组（交错）存储在内存中，例如：



In der Datei `cg2application.cpp` finden Sie eine Struktur 文件cg2application.cpp包含一个结构

```
struct MyVertex {
    GLfloat position[3];
    GLubyte color[3];
};
```

每个代表三角形的顶点。数组MyVertex cube\_data [36]包含绘制立方体所需的所有顶点。  
welche jeweils einen Vertex des Dreiecks repräsentiert. Das Array `MyVertex cube_data[36]` beinhaltet alle Vertices, die zum Zeichnen eines Würfels benötigt werden.

将整个cube\_data数组一次复制到bn缓冲区并相应地调整属性指针。修改glDrawArrays调用，不仅渲染三个顶点，而且渲染多维数据集的所有36个顶点。不再需要缓冲区color\_cb。

Kopieren Sie das gesamte Array `cube_data` am Stück in den Buffer `bn` und passen Sie die Attribut-Pointer entsprechend an. Ändern Sie den `glDrawArrays` Aufruf so, dass nicht nur drei Vertices, sondern alle 36 Vertices des Würfels gerendert werden. Den Buffer `color_cb` wird jetzt nicht mehr benötigt.



Neben dem bekannten `sizeof`-Operator gibt es in C/C++ auch den (unbekannteren) `offsetof`-Operator, mit dem man den Byte-Offsets eines Members innerhalb einer Struktur oder Klasse ermitteln kann, also z.B. `offsetof(MyVertex, color)`.

除了众所周知的sizeof运算符之外，C / C ++还具有（鲜为人知的）offsetof运算符，该运算符可用于确定结构或类中成员的字节偏移，例如，offsetof(MyVertex, color)

## 4. Indiziertes Rendering (Indexed Rendering) 索引渲染

除了glDrawArrays()之外，OpenGL还允许使用glDrawElements()进行所谓的索引渲染。在这种情况下，只有实际上不同的顶点必须存储在数组中，并且可以由几个基元引用。

Neben `glDrawArrays()` erlaubt OpenGL auch sogenanntes Indexed Rendering mittels `glDrawElements()`. Dabei müssen nur die tatsächlich verschiedenen Vertices im Array gespeichert werden und können von mehreren Primitiven referenziert werden.<sup>2</sup>

修改cube\_data数组和bn缓冲区以仅存储8个不同的顶点。然后创建一个索引数组GLuint index\_data [],重绘原始的12个三角形。

- (a) Modifizieren Sie das Array `cube_data` und den Buffer `bn` so, dass nur noch die 8 tatsächlich verschiedenen Vertices gespeichert werden. Erstellen Sie dann ein Index-Array `GLuint index_data[]`, mit dem sich die ursprünglichen 12 Dreiecke wieder zeichnen lassen.
- (b) Erstellen Sie einen weiteren Buffer `ibn` und binden Sie das Bufferobject an `GL_ELEMENT_ARRAY_BUFFER`. Laden Sie schließlich das Index-Array hoch. Passen Sie den Drawcall in `render_one_frame()` so an, dass mit `glDrawElements()` unter Verwendung der Indices aus dem in `GL_ELEMENT_ARRAY_BUFFER` gebundenen Buffer gerendert wird.

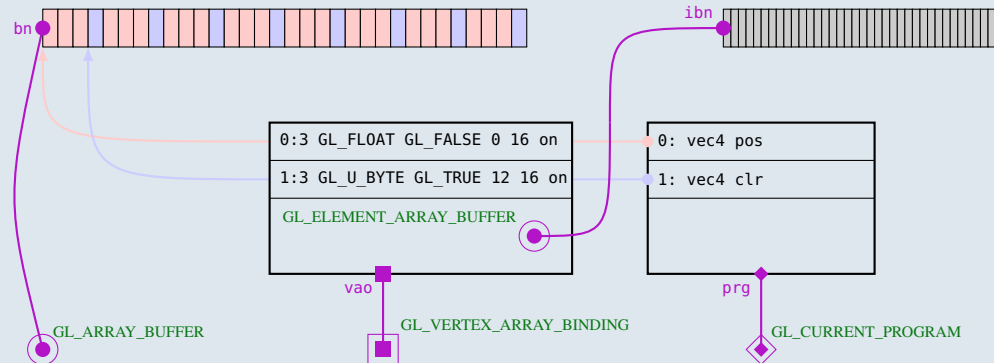
创建另一个缓冲区ibn并将缓冲区对象绑定到GL\_ELEMENT\_ARRAY\_BUFFER。最后，上传索引数组。调整render\_one\_frame()中的drawcall，使用glDrawElements()使用GL\_ELEMENT\_ARRAY\_BUFFER中绑定缓冲区的索引进行渲染。

<sup>2</sup>Damit kann die Größe der VBOs stark verringert werden. Im Durchschnitt ist ein Vertex in einem geschlossenen Dreiecksnetz Teil von sechs Dreiecken.

这可以大大减少VBO的大小。平均而言，封闭三角形网格中的顶点是六个三角形的一部分。

```
void glDrawElements(GLenum mode, GLsizei count, GLenum type, const GLvoid* indices)
```

Drawcall zum Indexed Rendering. Rendert count viele Vertices unter Verwendung der an GL\_ELEMENT\_ARRAY\_BUFFER gebundenen Indices als mode (GL\_POINTS, GL\_LINES, GL\_TRIANGLES, ...) Primitive. Der Parameter type bezieht sich dabei auf den Index-Typ (GL\_UNSIGNED\_INT, GL\_UNSIGNED\_SHORT, GL\_UNSIGNED\_BYTE) und der Parameter indices gibt einen Offset in Bytes relativ zum Anfang des Index-Buffer.



Der Binding-Point GL\_ELEMENT\_ARRAY\_BUFFER ist Teil des Vertex Array states. (Das Vertex Array speichert, aus welchem Buffer die Vertex Indices gelesen werden sollen.

绑定点GL\_ELEMENT\_ARRAY\_BUFFER是顶点阵列状态的一部分。（顶点数组存储应从哪个缓冲区读取顶点索引。