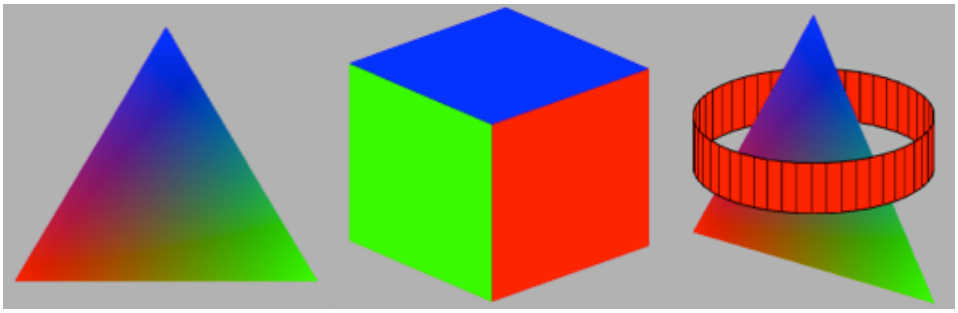
Computergrafik 2: Aufgabe 2.2

3D-Geometrie und Vertex Buffer Objects (VBO)



Lernziele / Motivation

In dieser Übung machen Sie sich mit der Darstellung von Geometrie durch Punkte, Linien und Dreiecke in WebGL vertraut, trainieren ein wenig Ihr räumliches Vorstellungsvermögen und üben den Umgang mit *Vertex Buffer Objects, Attribute Buffers und Element Buffers*.

Aufgabe 2.2.1: `Triangle` und Per-Vertex-Attribute

Studieren Sie das Modul `models/triangle.js`. Es besteht aus einem Konstruktor, der die drei Punkte eines Dreiecks in ein Vertex Buffer Object (VBO) schreibt, und einer `draw()`-Methode, die dieses VBO mittels `drawArrays()` als einzelne Punkte darstellt.

Ändern Sie dieses Modul so, dass das Dreieck als eine Fläche bestehend aus einem Dreieck gezeichnet wird. (Tipp: dies ist eine sehr kleine Änderung!).

Bisher wird das Dreieck in `Scene.draw()` mittels des GPU-Programs `programs.red` gezeichnet, welches jeden Vertex rot einfärbt. Nun soll das Dreieck mit einer von Ihnen spezifizierten *Farbe pro Vertex* dargestellt werden.

Dafür verwenden Sie zum Zeichnen des Dreiecks das GPU-Programm `programs.vertexColor` anstelle des Programm `programs.red`. Studieren Sie den zugehörigen Vertex-Shader `shaders/vertexColor.vs` - hier wird ein Vertex-Attribut namens `vertexColor` erwartet. Fügen Sie in `Triangle` also einen weiteren Attribut-Buffer (`vbo.Attribute`) hinzu, der eine Farbe pro Vertex enthält, binden Sie diesen Buffer an die Shader-Variable `"vertexColor"`.

Beachten Sie dabei, wie viele Farbkanäle pro Vertex der Farb-Buffer haben muss (RGB oder RGBA?). Dies können Sie leicht dem Vertex Shader entnehmen.

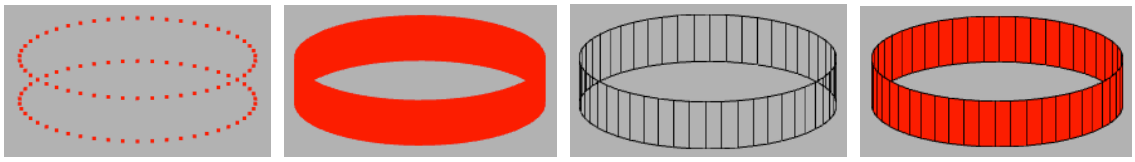
Aufgabe 2.2.2: `Cube`, Index Buffers und `drawElements()`

Studieren Sie das Modul `models/cube.js`. Es modelliert die sechs Seiten eines Würfels jeweils mittels separater Vertices (d.h. ein Vertex wird für jede Seite noch einmal wiederholt und existiert somit in jeweils drei Instanzen). Die Darstellung erfolgt jedoch lediglich als Punktewolke.

Erweitern Sie das Modul `cube.js` so, dass die Oberfläche des Würfels mit Hilfe von Dreiecken gezeichnet wird. Hierzu benötigen Sie einen zusätzlichen Index-Buffer (`vbo.Indices`), in dem jeweils die richtigen Vertices zu einem Dreieck verbunden werden (z.B. besteht die Vorderseite aus den beiden Dreiecken ABC und ACD). In der `draw()`-Methode des Cube muss der `drawArrays()`-Aufruf durch einen `drawElements()`-Befehl ersetzt werden. Die Details zu `vbo.Indices` und zu den Zeichenbefehlen finden Sie im Handout zur SU. Objekte vom Typ `vbo.Indices` bieten verfügen über eine Methode `numIndices()`, welche die Anzahl der Indizes im Buffer zurückliefert.

Fügen Sie dem `Cube` Per-Vertex-Farben hinzu, so dass jede *Seite* des Würfels mit einer eigenen konstanten Farbe dargestellt wird (z.B. oben und unten rot, links und rechts grün, vorne und hinten blau). Verfahren Sie hierbei analog zu `Triangle`.

Aufgabe 2.1.3: Oberfläche vs. Wireframe



Erweitern Sie bitte das Modul `models/band.js`, welches ein "Band" oder einen "Ring" darstellen soll. Das Band wird in einer `for`-Schleife durch Vertices konstruiert, die auf zwei parallelen Kreisen liegen. Implementieren Sie analog zum Würfel die Darstellung der Oberfläche mittels Dreiecken. Die ersten beiden Dreiecke des Bandes haben z.B. die Indizes [0, 1, 2, 2, 1, 3]. Verwenden Sie das Attribut `config.drawStyle` des Konstruktors von Band, so dass man beim Erzeugen eines Band-Objekts angeben kann, ob das Band als Punktewolke oder mittels Flächen dargestellt werden soll.

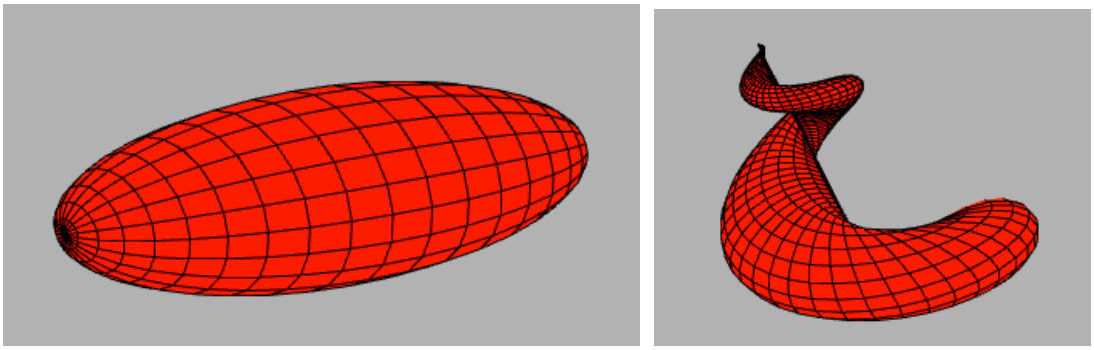
Implementieren Sie zusätzlich eine Wireframe-Darstellung, bei welcher der Ring so dargestellt wird, als sei er aus Linien zu Rechtecken zusammengesetzt. Je nach Wert von `drawStyle` soll das Band also als Punkte, Linien oder Flächen gezeichnet werden. Erlauben Sie mittels der `drawOptions` in `scene.js` dem Benutzer, zwischen Wireframe- und Oberflächendarstellung zu wählen (verwenden Sie dazu einfach zwei getrennte `Band`-Objekte und zwei `drawOptions` "Solid Band" und "Wireframe Band" o.ä.).

Stellen Sie das Wireframe-Objekt in einer anderen Farbe, z.B. schwarz dar. Erzeugen Sie hierzu im Konstruktor der Szene analog zu `programs.red` ein weiteres GPU-

Programm `programs.uni` basierend auf den Shadern `unicolor.vs` und `unicolor.fs`. Studieren Sie den Fragment-Shader - dieser erwartet, dass die zu verwendende Farbe als eine uniform-Variable `uniColor` vom Typ `vec4` definiert wird.

Setzen Sie also, bevor Sie das Wireframe-Band zeichnen, im `draw()` der Szene diese Uniform-Variable in `programs.uni` auf die von Ihnen gewählte Farbe, und zeichnen Sie das Wireframe-Objekt mit Hilfe dieses GPU-Programms.

Aufgabe 2.2.4: Fläche und Wireframe für `ParamericCurve`



Setzen Sie die Darstellung des Wireframes sowie die flächenhafte Darstellung auch für das Objekt `ParametricSurface` um, so dass Ihre parametrischen Flächen als Oberflächen mit Gitterlinien dargestellt werden.

Noch ein Tipp: wahrscheinlich wird bei Ihnen die kombinierte Darstellung von Linien und Flächen bei der Animation "flackern". Recherchieren Sie hierzu den Befehl `gl.PolygonOffset()` und wenden Sie ihn an, um das sogenannte "Z Fighting" zu unterdrücken.

Abgabe

Diese Aufgaben sind der erste von mehreren Teilen der Aufgabe 2. Die Abgabe der gesamten Aufgabe 2 soll via Moodle bis zu dem dort angegebenen Termin erfolgen. Verspätete Abgaben werden wie in den Handouts beschrieben mit einem Abschlag von 2/3-Note je angefangener Woche Verspätung belegt. Geben Sie bitte pro Gruppe jeweils nur eine einzige `.zip`-Datei mit den Quellen Ihrer Lösung sowie mit den ggf. geforderten Screenshots ab.

Demonstrieren und erläutern Sie dem Übungsleiter Ihre Lösung *in der nächsten Übung nach dem Abgabetag*. Die Qualität Ihrer Demonstration ist, neben dem abgegebenen Code, ausschlaggebend für die Bewertung! Es wird erwartet, dass alle Mitglieder einer Gruppe anwesend sind und Fragen beantworten können.