Media Informatics /
Graphics & Geometric Computing

Prof. Dr. Marcel Campen
Steffen Hinderink

UNIVERSITÄT OSNABRÜCK

Computergrafik (SoSe 2023)

Blatt 3

fällig Sonntag **14. Mai**, 24:00

Verspätet eingereichte Abgaben können nicht berücksichtigt werden.

Aufgabe 1 (Theorie: Frustums, Linien, Dreiecke)

(a) (4P) Frustum-Matrix:

1. Geben Sie die Frustum-Matrix für einen horizontalen Öffnungswinkel (*fovx*) von $90°$ und einen vertikalen Öffnungswinkel (*fovy*) von $60°$ mit near-Plane $n = 21$ und far-Plane $f = 6000$ an.
2. Das entsprechende Frustum (der Pyramidenstumpf, der den sichtbaren Bereich vor der Kamera begrenzt) hat acht Eckpunkte. Geben Sie die Koordinaten von zwei (beliebig ausgewählten) dieser Eckpunkte aus (a) an, davon jedoch ein Punkt in der near-Plane, ein Punkt in der far-Plane.
3. Wenden Sie die Frustum-Matrix aus (a) auf diese beiden Punkte aus (b) an und konktrollieren Sie, ob diese (ggf. nach Dehomogenisierung) wie erwartet Ecken des Würfels $[-1, 1] \times [-1, 1] \times [-1, 1]$ bilden.

(b) (2P) Der Punkt $(11, -8)^\mathsf{T}$ liegt auf der Gerade durch $(18, -14)^\mathsf{T}$ und $(-3, 4)^\mathsf{T}$. Stellen Sie ihn als Affinkombination dieser beiden Punkte dar.

(c) (2P) Berechnen Sie die baryzentrischen Koordinaten des Punktes $(1, -7)^\mathsf{T}$ bezüglich des Dreiecks mit den Ecken $a = (0, 0)^\mathsf{T}$, $b = (1, 5)^\mathsf{T}$ und $c = (-4, 4)^\mathsf{T}$.

(d) (2P) Markieren Sie in einem Pixelraster die Pixel, die nach den Regeln des Bresenham-Algorithmus gefärbt werden, wenn eine Linie vom Punkt $(24, -9)^\mathsf{T}$ zum Punkt $(27, 1)^\mathsf{T}$ gezeichnet wird. Die Koordinaten der Pixelzentren sind dabei ganze Zahlen.

Aufgabe 2 (Praxis: Linien und Dreiecke)

Hinweis: In JavaScript können Sie sich Hilfsfunktionen innerhalb der vorgegebenen Funktionen definieren, um Ihren Code ggf. eleganter zu gestalten.

Hinweis: Sie können die Größe des Canvas mit der Konstanten `size` verkleinern, falls die Darstellung auf Ihrem Rechner zu ruckelig ist.

(a) 1. (1P) Vervollständigen Sie die Funktion `createLines`, die ein Array von Linien zurückgeben soll. Erstellen Sie mehrere Liniensegmente, so dass diese einen beliebigen dreidimensionalen Körper (z.B. einen Würfel) formen. Zur Verwaltung der Linien wird Ihnen die Klasse `Line` zur Verfügung gestellt. Linien werden spezifiziert durch einen Start- und einen Endpunkt. Für Punkte wird die bekannte `Point`-Klasse verwendet. Eine Linie erstellen Sie durch `new Line(a, b)`. Die Koordinaten der Punkte sollten zwischen -100 und +100 liegen, damit sie im Sichtfeld der Kamera sind. Anschließend sollten Ihnen noch nicht die Linien, jedoch bereits ihre Endpunkte angezeigt werden (aufgrund des Codes, der mit `Test a` markiert ist). Per Maus kann die Szene rotiert werden.

2. (4P) Vervollständigen Sie die Funktion `drawLine`, welche vom vorhandenen Code für jede Linie nach Transformation, Projektion und Viewport-Transform aufgerufen wird. Zeichnen Sie eine Linie, indem Sie den Bresenham-Algorithmus (oder eine vereinfachte etwas weniger effiziente Variante, wie in der Vorlesung vorgestellt) implementieren. Achten Sie darauf, zunächst die Richtung und Steigung der Linie zu bestimmen und entsprechend darauf zu reagieren. Start- und Endpunkt der Linie stehen in den Membervariablen `a` und `b` zur Verfügung. Das Pixel (x, y) färben Sie schwarz, indem Sie `setPixel(x, y)` aufrufen. Wenn die Checkbox "Zeichne Linien" aktiv ist, sollten Sie nun die Linien sehen.

(b) 1. (1P) Vervollständigen Sie die Funktion `createTriangles`, die ein Array von Dreiecken zurückgeben soll. Erstellen Sie mehrere Dreiecke, so dass diese einen beliebigen dreidimensionalen Körper (z.B. einen Würfel) formen. Zur Verwaltung der Dreiecke wird Ihnen die Klasse `Triangle` zur Verfügung gestellt. Ein Dreieck erstellen Sie durch `new Triangle(a, b, c, color)`. Jedes Dreieck besteht aus den drei Eckpunkten vom Typ `Point` sowie einem Array `color` mit drei Einträgen [r,g,b] für die Farbinformationen. Auf diese Weise können Sie jedem Dreieck eine Farbe zuweisen, um diese später in der Darstellung besser auseinanderhalten zu können. Die Koordinaten der Punkte sollten wieder zwischen -100 und +100 liegen, die Farbwerte zwischen 0 und 255. Nach diesem Teil sollten die Eckpunkte, jedoch noch nicht die Flächen der Dreiecke angezeigt werden (aufgrund des Codes, der mit `Test b` markiert ist).

2. (4P) Vervollständigen Sie die Funktion `drawTriangle`, welche vom vorhandenen Code für jedes Dreieck nach Transformation, Projektion und Viewport-Transform aufgerufen wird. Zeichnen Sie ein Dreieck, indem Sie die Pixel, deren Mittelpunkt innerhalb des Dreiecks liegt mit der Farbe des Dreiecks einfärben. Nutzen Sie dazu `setPixel(x, y, color)`. Testen Sie nicht naiv für jedes Pixel des Canvas, ob es innerhalb des Dreiecks liegt, sondern beschränken Sie den Test auf die Pixel innerhalb einer "BoundingBox" des Dreiecks. Wenn die Checkbox "Zeichne Dreiecke" aktiv ist, sollten Sie nun die Dreiecke sehen.

   Hinweis: Werden mehrere Dreiecke auf die gleiche Stelle projiziert, sehen Sie dort ggf. nicht das vorderste, sondern das zuletzt gezeichnete. Die Dreiecke werden anhand ihres Mittelpunktes von hinten nach vorne sortiert, bevor Ihre Funktion `drawTriangle` in dieser Reihenfolge aufgerufen wird. Dies schafft (teilweise) Abhilfe – vollständig korrekte Lösungen für dieses Problem schauen wir uns später in der Vorlesung an.

Aufgabe 3 (Bonus: Antialiasing von Linien)

Modifizieren Sie den Code so, dass die Linien mit Antialiasing (mittels eine Variante Ihrer Wahl) gezeichnet werden, wenn die Checkbox "Antialiasing" aktiv ist. Die Variable `antialiasing` enthält den Wert der Checkbox. Damit Pixel teiltransparent eingefärbt werden können (so dass auch mehrere übereinander gezeichnete Linien korrekt erscheinen) muss ggf. die Funktion `setPixel` erweitert werden.