Media Informatics /
Graphics & Geometric Computing

Prof. Dr. Marcel Campen
Steffen Hinderink

UNIVERSITÄT OSNABRÜCK

Computergrafik (SS 2020)

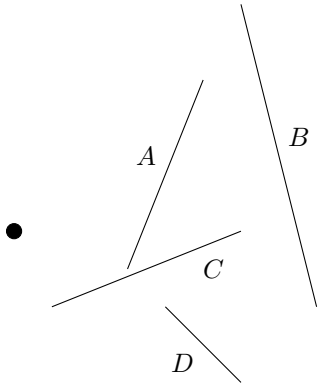
Blatt 5

fällig Sonntag **31. Mai**, 24:00

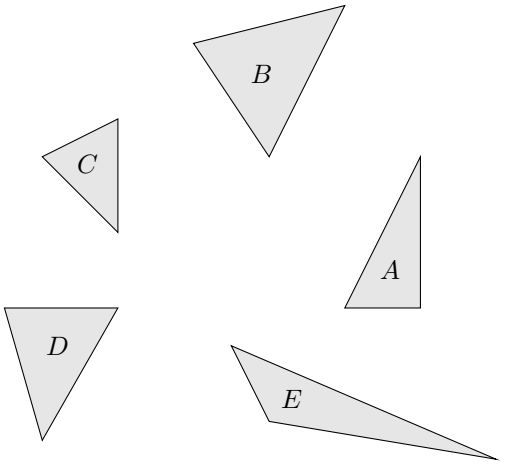
Verspätet eingereichte Abgaben können nicht berücksichtigt werden.

Aufgabe 1 (Theorie: Sichtbarkeit)

(a) (2P) Wir stellen uns eine 2D Variante des 3D Painter's Algorithmus aus der Vorlesung vor, mit Geradensegmenten und ihren Stützgeraden statt Dreiecken und ihren Stützebenen. Bestimmen Sie, in welcher Reihenfolge die abgebildeten Geradensegmente von diesem 2D Painter's Algorithmus gezeichnet werden. Der Betrachter befindet sich am schwarzen Punkt. Begründen Sie für jedes Segment, aufgrund welcher Regel es vor oder nach einem anderen gezeichnet wird.



(b) (3P) Konstruieren Sie einen so-balanciert-wie-möglichen BSP-Baum für die abgebildeten fünf Objekte, so dass jedes Blatt genau 1 Objekt enthält. Verwenden Sie eine beliebige Split-Regel – es ist nicht nötig, die verwendete Regel oder die Ebenengleichungen hier anzugeben. Zeichnen Sie eine grafische Darstellung der durch den BSP-Baum beschriebenen Partitionierung des Raumes in die Abbildung ein, und zeichnen Sie eine abstrakte Darstellung des Baumes an sich.



(c) (2P) Berechnen Sie den Schnittpunkt des Strahls von Punkt $(4, 1, 3)$ in Richtung $(-\frac{1}{\sqrt{2}}, 0, -\frac{1}{\sqrt{2}})$ mit dem Dreieck mit den Eckpunkten $(0, 0, 0)$, $(2, 0, 0)$, $(0, 2, -1)$. Geben Sie Ihre Rechenschritte an.

(d) (2P) Auf ein Pixel treffen Fragmente mit den folgenden Werten (in dieser Reihenfolge):

- $[z = 0.3;\ rgb = \#000000]$
- $[z = 0.0;\ rgb = \#BABE00]$
- $[z = -0.7;\ rgb = \#00FFCC]$
- $[z = 1.0;\ rgb = \#1337DD]$
- $[z = -0.1;\ rgb = \#0FE1FE]$

Eine sechsstellige hexadezimale Zahl steht hierbei für eine Farbe. Dabei geben je zwei hexadezimale Ziffern (1 Byte) den roten, grünen und blauen Farbanteil an. Die r- g- und b-Werte können also zwischen $0 = 00_{\text{hex}}$ und $255 = FF_{\text{hex}}$ liegen. Geben Sie den Zustand des Color-Buffer und des z-Buffer an der Stelle dieses Pixels nach Verarbeitung eines jeden einzelnen Fragments (das heißt insgesamt 5 Zustandspaare) an.

Hinweis: Wir nehmen hier an, dass Normalized Device Coordinates (NDC) verwendet werden, die near-Plane also bei $z = -1$ und die far-Plane bei $z = 1$ liegt.

(e) (1P) Auf ein Pixel treffen Fragmente mit den folgenden Werten (in dieser Reihenfolge):

- $[z = -0.9;\ rgb = \#00FFFF,\ \alpha = 0.8]$
- $[z = 0.1;\ rgb = \#010101,\ \alpha = 0.2]$
- $[z = 0.5;\ rgb = \#ABCDEF,\ \alpha = 0.5]$
- $[z = -0.5;\ rgb = \#00FF00,\ \alpha = 1.0]$
- $[z = -0.1;\ rgb = \#FFFFFF,\ \alpha = 0.7]$

Geben Sie den Zustand des α-Buffer nach Verarbeitung aller (nicht eines jeden einzelnen) Fragmente an und berechnen Sie daraus die finale Farbe des Pixels, unter der Annahme, dass der α-Buffer Algorithmus aus der Vorlesung verwendet wird.

Aufgabe 2 (Praxis: z-Buffer)

(a) (4P) Vervollständigen Sie die Funktion `resetBuffers`, die den `zBuffer` und `colorBuffer` mit initialen Werten füllen soll. Die Funktion wird vor dem Rendern eines jeden Bildes aufgerufen. Die Buffer sind zweidimensionale Arrays mit `size`×`size` vielen Einträgen.

(b) (6P) Vervollständigen Sie die Funktion `updateBuffers`, in der für das Pixel in der Zeile `row` und der Spalte `col` der z-Wert `z` und die Farbe `color` verarbeitet werden sollen. Abhängig vom z-Wert müssen Color- und z-Buffer aktualisiert werden.

Hinweis: Nach der Frustum-Transformation gilt $z = 1$ für Punkte auf der far-Plane und $z = -1$ für Punkte auf der near-Plane.

Aufgabe 3 (Bonus: α-Buffer)

Ersetzen Sie den z-Buffer aus Aufgabe 2 durch einen α-Buffer. Sortieren Sie am Ende die Elemente eines jeden Pixels und bestimmen Sie die finale Farbe. Fügen Sie jedem Dreieck einen α-Wert hinzu, der die Opazität angibt. Stellen Sie eine zweite Instanz der Kugel (oder ein anderes Objekt) in anderer Farbe an anderer Stelle dar, so dass die Teiltransparenz besonders deutlich wird.