Bildbasierte Modellierung SS 2018
Übungsblatt 7

TU Braunschweig
Prof. Dr.-Ing. Marcus Magnor
Institut für Computergraphik

JP Tauscher
tauscher@cg.cs.tu-bs.de

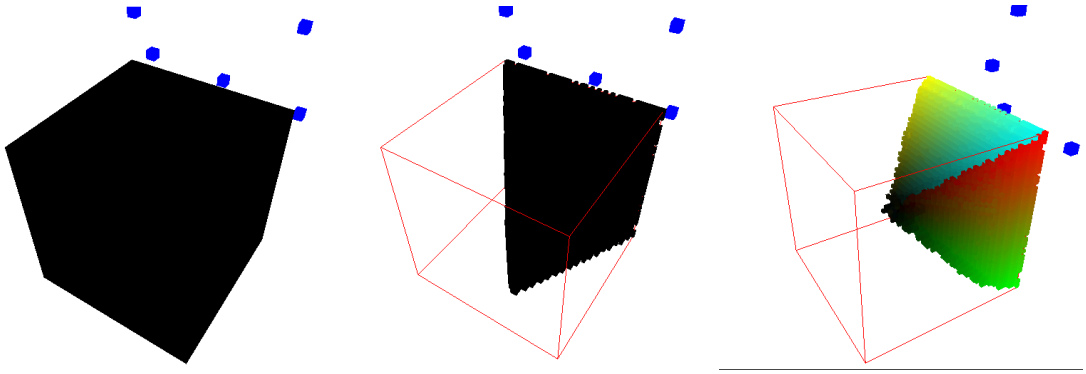
5.6.2018

Abgabe: Präsentation der bearbeiteten Aufgaben in der Übung am 12.6.2018.

Für die Programmieraufgaben kann in Gruppen von max. 3 Leuten zusammengearbeitet werden. Dabei muss aber jeder einzelne in der Lage sein, alle Teile des Programms zu erklären. Die Materialien für die Programmieraufgaben sind jeweils erhältlich unter:

`https://graphics.tu-bs.de/teaching/ss17/bbm`

Das Ziel dieses Übungsblatts ist die Implementierung des *Space-Carving*-Algorithmus, um aus Fotografien eines Objekts eine Volumendarstellung zu erhalten. Der Algorithmus wurde ursprünglich von Kutulakos und Seitz in *A Theory of Shape by Space Carving* veröffentlicht.
Zwei Beispielszenen, *scene1* und *scene2*, sind als Archive angefügt und müssen zunächst entpackt werden. Das Programm kann mit dem Aufruf `./ex07 <scenename> <outfile> <color_threshold> <grid_resolution>` gestartet werden, wobei `<scenename>` der Name der Szene, also *scene1* oder *scene2*, `<outfile>` eine .obj-Datei für den Output, `<color_threshold>` ein Schwellwert zwischen 0 und 255 und `<grid_resolution>` die Auflösung des Voxelgitters ist. Der Befehl `./ex07 scene1 out.obj 100 50` lädt also alle benötigten Daten aus dem Ordner *scene1*, benutzt einen Farb-Schwellwert von 100, arbeitet auf einem $50 \times 50 \times 50$ Voxelgitter und speichert das Ergebnis in out.obj. Die .obj-Datei kann mit allen gängigen Mesh-Viewern geöffnet werden, die farbige Vertices unterstützen. Im CIP-Pool ist das Program *MeshLab* installiert. Zu Anfang sollte ein schwarzer Würfel und kleine blaue Würfel an den Kamerapositionen angezeigt werden. Die Datei ist zu Anfang recht groß und läd unter Umständen einen Moment.



Von allen möglicherweise sichtbaren Voxeln im Gitter (linkes Bild) gehören nur Voxel zum Objekt, die nicht auf weißen Bildhintergrund projiziert werden (mittleres Bild). Vergleicht man Farbwerte verschiedener Kameras, lässt sich 3D-Struktur rekonstruieren (rechtes Bild).
Die Voxelmittelpunkte werden immer in einen Einheitswürfel von $(-0.5, -0.5, -0.5)$ bis zu $(0.5, 0.5, 0.5)$ gesetzt, d.h. der Mittelpunkt von Voxel $(0, 0, 0)$ wird an die Stelle $(-0.5, -0.5, -0.5)$, der Mittelpunkt von Voxel ($\texttt{grid\_resolution} - 1, \texttt{grid\_resolution} - 1, \texttt{grid\_resolution} - 1$) an die Stelle $(0.5, 0.5, 0.5)$ gesetzt.

Die eingebundene Bibliothek `glm` enthält eine Reihe von hilfreichen Klassen (`vec3`, `vec4`, `mat3`, `mat4`), die sehr intuitiv in der Benutzung sind. So kann ein `vec4 p` mit einer 4×4 Matrix `mat4 m` multiplziert werden, indem `vec4 mp = m p` aufgerufen wird.

7.1 Bildkonsistenz (5 Punkte)

Iteriere über alle Voxel, indem du zuerst alle Voxel einer Ebene mit gleicher z-Koordinate betrachtest und dann die Ebene entlang der positiven z-Achse verschiebst. Benutze `textureMatrices`, um die Voxelmittelpunkte in die einzelnen Bilder zu projizieren. Ist die berechnete Position innerhalb der Bildes, behalte sie, ansonsten wird dieser Voxel aus dem Gitter entfernt. Dies führt zu einer Menge von Voxeln, die möglicherweise (abhängig vom Objekt in der Szene) in allen Bildern sichtbar sind. Setzt man alle Voxel, die diese Bedingung erfüllen schwarz, so sollte das Ergebnis aussehen wie im linken Bild.

7.2 Silhouettenkonsistenz (5 Punkte)

Es wird eine weitere Bedingung hinzugefügt: Alle gegebenen Bilder haben einen weißen Hintergrund. Wenn also ein Voxel in irgendeinem der Bilder auf einen weißen Pixel projiziert wird, wird er aus dem Voxelgitter entfernt. Nimmt man die Bildkonsistenz hinzu, sollte das Ergebnis aussehen wie im mittleren Bild.

7.3 Space Carving (10 Punkte)

- Erweitere das bisher entstandene Framework um eine weitere Bedingung: Vergleiche die Farbe eines Voxels in allen Eingabebildern. Nur wenn sich die Farben paarweise um weniger als `diffThresh` pro Farbkanal unterscheiden, werden sie im Gitter belassen. Weise diesen Voxeln die mittlere Farbe aller Pixelfarben zu. Die 3D-Szene wird in einer .obj-Datei gespeichert. Farbwerte liegen dort im Bereich $[0, 1]$, sie müssen also entsprechend umgewandelt werden.

- Aus numerischen Gründen vergleiche nun nicht die Farbwerte aller Eingabebilder, sondern nur der 3 Bilder, die am nächsten am Mittelwert liegen. Weise dem Voxel den Mittelwert dieser 3 Farbwerte zu. Eine `priority_queue` und `pair` können die Arbeit hier erleichtern.

- Erweitere das Programm so, dass es mehrere Durchläufe (*sweeps*) benutzt: Betrachte nicht nur eine Ebene, die sich entlang der positiven z-Achse bewegt, sondern auch Ebenen entlang der positiven x- und y-Achse und entlang der negativen Achsen.

- Ziehe nun zum Farbvergleich nur Kameras hinzu, die sich in Bezug auf die Bewegungsrichtung hinter der Ebene befinden.