BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Prof. Dr.-Ing. Kristian Hildebrand

Computergrafik 2 / Aufgabe 1.2
kd-Baum

SoSo 2016

In dieser Aufgabe implementieren Sie den Aufbau eines kd-Baumes und messen den Geschwindigkeitsunterschied zwischen einer linearen Suche über ein Array und einer Nächsten-Nachbarn-Suche über einem kd-Baum.

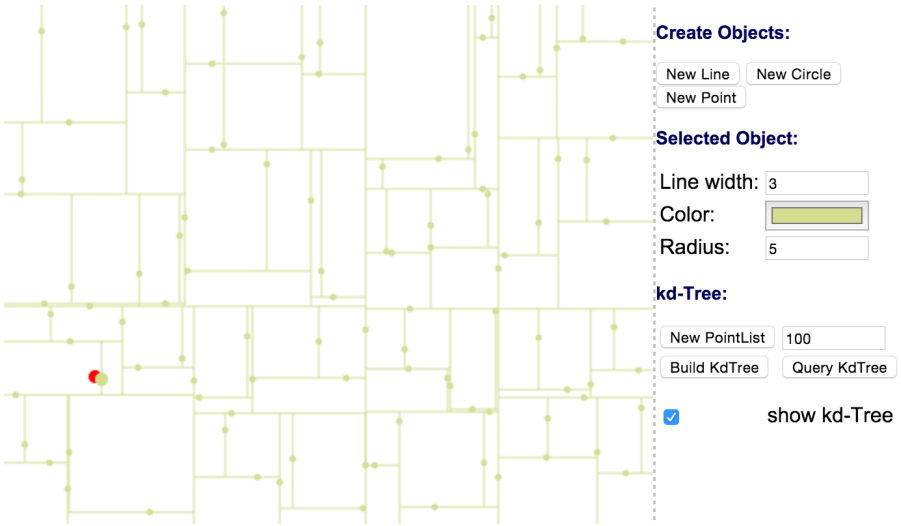

Abbildung 1: Ergebnisausgabe

Vorbereitung Laden Sie *cg2-a01.2-kdtree.zip* von Moodle herunter und entpacken Sie die Datei. Das Paket enthält *HtmlController* mit neuen Funktionen, die Sie sich in Ihren Sourcecode übernehmen sollen und einen Ordner *kdtree*. Fügen Sie den Ordner in ihren bisherigen Aufgabenordner hinzu.

Implementierungshinweise Wenden Sie ihre bisherigen Javascript Kenntnisse an. Es ist vieles an notwendigem Sourcecode vorgegeben. Schauen Sie sich den Code genau durch und verstehen Sie wie dieser funkioniert (z.b. *findNearestNeighbor*). Sollten Sie Schwierigkeiten haben, den bestehenden Code in der vorgegebenen Version zu benutzen und möchten den Aufbau des Baumes auf eine andere Art implementieren steht Ihnen das natürlich frei und ist durchaus gewünscht. Dies sind **keine starren Vorgabe**.

Aufgabe 1: Einfügen und Anpassen des Übungs-Sourcecodes

1 Fügen Sie alle neue Module (Dateien) in *app.js* dem *requirejs.config*-Objekt hinzu.

2 Ändern Sie den *HtmlController* so, dass er zwei weitere Objektvariablen bekommt: *pointList* and *kdTree*. Diese können zunächst uninitialisiert sein, müssen dann aber in den *onButtonClick*-Callback Funktionen verwendet und initialisiert bzw. gefüllt werden.

```
1 var HtmlController = function(context,scene,sceneController) {
2     var kdTree;
```

```
3     var pointList = [];
```

3 Übernehmen Sie aus dem *HtmlController* die *click*-Handler, die an folgende DOM ids geknüpft sind: *btnNewPointList*, *visKdTree*, *btnBuildKdTree*, *btnQueryKdTree*. Binden Sie die click-Handler in Ihre *index.html* ein indem Sie die dazugehörigen HTML Buttons und weiteren Interfaceelemente hinzufügen. Orientieren Sie sich am obigen Ergebnis-Screenshot.

Aufgabe 2: Aufbau des kd-Baumes

Das *kdtree* Module enthält bereits Funktionen zum Aufbau des Baumes und zum Finden des nächsten Nachbarn zu einem gegebenen Anfragepunkt. Die *build*-Funktion ist jedoch leer. Implementieren Sie den Aufbau des Kd-Baumes und nutzen dafür gegebene Funktionen aus *kdutil*.

Nach einem erfolgreichen Aufbau des kd-Baumes sollten Sie:

a) Eine Liste von Punkten erzeugen können.

b) Die Punktliste erfolgreich in den Baum einfügen können.

c) Den *btnQueryKdTree*-Handler nutzen können, um einen Query-Punkt zu erzeugen. Dieser Punkt wird in der *btnQueryKdTree*-Handler Funktion genutzt, um die kd-Nächste-Nachbarn-Suche durchzuführen.

d) Eine korrekte Visualisierung des kd-Baumes sehen können. Nehmen Sie sich die Zeit zu verstehen, ob die Visualisierung korrekt ist und wo/wann eventuell Fehler im Baum sind.

Implementierungshinweise Arbeiten Sie beim Entwickeln unbedingt mit der Entwicklerkonsole und dem Debugger. Nutzen Sie zu Beginn nur eine kleine Anzahl von Punkten < 10. Setzen Sie einen Debug-Haltepunkt in der Visualisierungsfunktion. So können Sie **nach dem Aufbau des Baumes** überprüfen, welche Knoten wie im Baum eingefügt wurden.

Aufgabe 3: (Zusatzaufgabe) Geschwindigkeitsmessung

Notwendig, um 1.0 erzielen zu können Implementieren Sie eine lineare Suche über die Punktliste in die leere Funktion *kdutil.linearSearch*. Das Ergebnis beider Suchen (kd-Nächste-Nachbarn-Suche und lineare Suche) muss gleich sein. Messen Sie im Javascript die Ausführungszeiten beider Suchmethoden über eine steigende Anzahl von Punkten und geben diese in Form einer kleinen Tabelle (von Hand, Excel o.ä.) aus. Ab welcher Punktanzahl sind Performanceunterschiede der beiden Suchmethoden erkennbar? **Hinweis: Schalten Sie hierfür die Visualisierung aus.**

Bewertungskriterien Die Bewertung erfolgt auf Basis Ihrer Demonstration in der Übung nach der Abgabe, sowie ggf. zusätzlich auf Basis der Inspektion Ihres Quellcodes. Die Qualität Ihrer Präsentation sowie der Erklärungen Ihres Codes sind ausschlaggebend.

Wenn Sie alle Pflichtaufgaben korrekt implementiert haben, Ihr Code gut strukturiert ist und Sie ihn gut erklären können, erhalten Sie die Note 2.0.

Wenn Sie zusätzlich auch die optionalen Aufgaben implementiert haben und Ihr demonstriertes Codeverständnis, auch für das Framework, exzellent ist, können Sie die Note 1.0 erreichen.

Abgabe Dies ist *Teilaufgabe 1.2*; die Bearbeitungszeit der Teilaufgabe ist für ca. eine Woche ausgelegt, kann aber aufgrund der Anfangshürden etwas mehr Zeit benötigen. Die Abgabe der gesamten Aufgabe 1 soll via Git bis zu dem dort angegebenen Termin erfolgen. Der letzte Commit/Push zählt. Lokale Änderungen am SourceCode nach der Deadline sind nicht erlaubt. Verspätete Abgaben werden mit einem Abschlag von 1.0-Notenpunkten je angefangener Woche Verspätung belegt.

Demonstrieren und erläutern Sie 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.