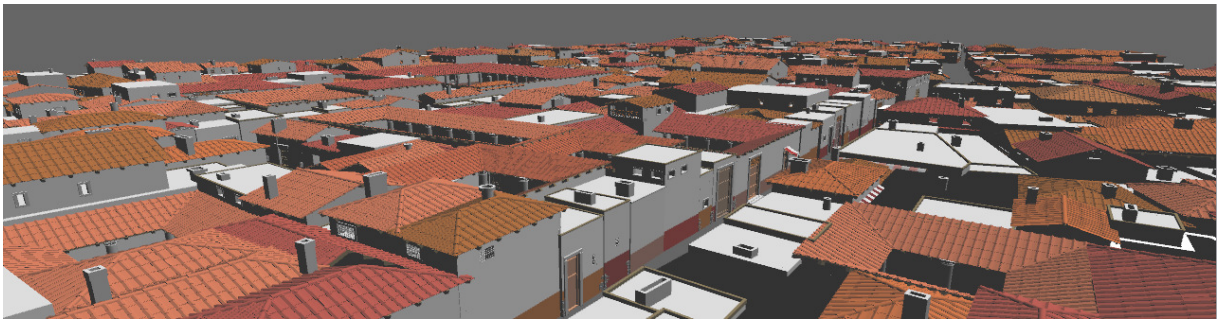


Out of core Projektpraktikum

Institut für Computergraphik und Algorithmen – TU Wien

Alex Druml - 0625181
Markus Ernst - 0625316
Lukas Rössler - 0625652

1. Allgemeines

Dieses Projekt befasst sich mit dem Echtzeit-Rendern von Modellen die aufgrund ihrer Größe nicht komplett im RAM bzw. VRAM gehalten werden können. Die Entwicklung solcher Out-of-Core Algorithmen wird durch die immer weiter steigende Anzahl an Polygonen detaillierter Szenen erforderlich. Mögliche Quellen derartiger Modelle sind Konstruktion von z.B. Flug- und Fahrzeugen oder Gebäuden, aber auch Messungen mittels 3D Scannern produzieren solch große Datenmengen. Das Darstellen dieser großen Datensätze in Echtzeit ist vor allem für interaktive Anwendungen wichtig.

Das dabei verwendete CHC++ (Coherent Hierarchical Culling - (1)) Framework ermöglicht durch Konstruktion einer Bounding Volume Hierarchy (BVH) eine effiziente Sichtbarkeitsklassifikation mittels Occlusion-Queries. Diese Sichtbarkeitsinformationen und ein spezielles Modelformat dienen als Basis für unseren Fetching Algorithmus. Die Aufgabe des Fetching Algorithmus ist es, den verfügbaren RAM bzw. VRAM möglichst effizient auszunutzen, so dass möglichst viele der sichtbaren Objekte dargestellt werden können. Auch das Freigeben von Speicher (Löschen von nicht mehr sichtbaren Objekten) muss der Algorithmus übernehmen.

Unser Algorithmus benötigt dafür ein spezielles Modelformat ('.demv2'), das wir ebenfalls entwickelt haben. Als Basis diente das '.dem' Format, das die Engine bereits lesen konnte. Unsere Erweiterung beinhaltet die Aufteilung des Formats in ein Index File und mehrere Geometrie Files, damit die Anwendung beim Starten möglichst wenig (nur das Index File) laden muss. Das Nachladen (aus den Geometrie Files) übernimmt der Fetching Algorithmus.

Die Tatsache, dass dieses Modelformat benötigt wird, macht eine Konvertierung des Models notwendig, aber auch dafür wurde ein Tool entwickelt (siehe Abschnitt 'Converter').

2. Anwendungsdetails

Projekt einrichten und kompilieren

Um die Anwendung starten/kompilieren zu können sind folgende Punkte zu erfüllen:

- Microsoft Visual Studio 2008
- nVidia CG Framework (ins besonders müssen die Umgebungsvariablen CG_INC_PATH und CG_LIB_PATH korrekt gesetzt sein)(2)
- Im Projekt müssen einige Settings ergänzt werden, da Visual Studio diese in den .user files speichert:

- Beim Projekt "MainApp Out-of-Core" muss bei Properties → Debugging das Feld "Working Directory" auf "$(SolutionDir)dist" gesetzt werden. Außerdem ist es sinnvoll bei "Command Arguments" das gewünschte env File (z.B. Pompeii.env) einzutragen, damit die Anwendung von Visual Studio aus korrekt gestartet werden kann.

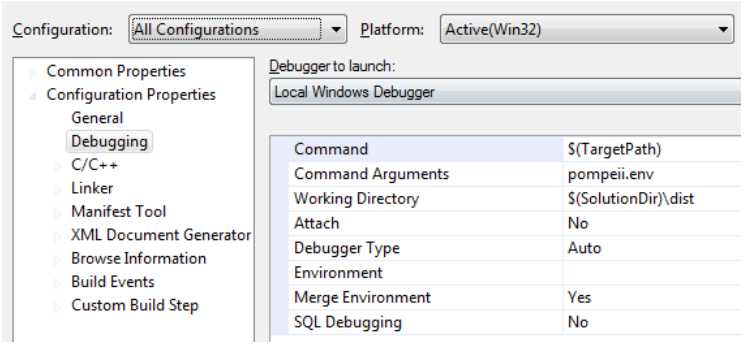


Abbildung 1 - Projektkonfiguration

Converter

Konvertiert *.obj, .ply, .dem, .demv2* auf *.dem* und *.demv2*

Am einfachsten ist der Converter verwendbar, indem einfach ein/mehrere Files auf die .exe gezogen werden, es können auch Verzeichnisse verwendet werden, in diesem Fall wird das Verzeichnis rekursiv traversiert und alle unterstützten Modelformate automatisch gelesen (können auch unterschiedliche Formate in einem Verzeichnis sein)

Einstellbare Optionen:

- bool **useDemv2**: falls true, wird auf .demv2 konvertiert, andernfalls auf .dem
- bool **flipFaces**: Invertiert die Reihenfolge der Vertices für jedes Face, falls true
- bool **useTextures**: falls false, werden Texturen und Texturkoordinaten beim Schreiben weggelassen
- bool **ignoreGroups**: falls true, werden (nur für .obj reader) group commands im .obj file ignoriert (erstellt eine große Gruppe / file, dadurch gehen natürlich Materialien verloren)
- bool **sceneEntitiesPerGroup**: (nur für .obj reader) falls true werden SceneEntities pro group command im .obj file erstellt, andernfalls wird eine SceneEntity pro .obj File erstellt (mit mehreren Shapes (eine für jede group im .obj file))
- int **coordinateSystem**[3]: Damit können Koordinatenachsen invertiert / vertauscht werden. {1,2,3} bedeutet (x=x, y=y, z=z), {1,-3,2} bedeutet (x=x, y=-z, z=y)
- float **rot[X|Y|Z]**: Rotationswinkel in Grad
- float **trans[X|Y|Z]:** Translation
- float **scale[X|Y|Z]:** Skalierung

Aus Rotation, Translation und Skalierung wird eine Transformationsmatrix aufgebaut (transform = scale*translation*rotationX*rotationY*rotationZ) die auf jeden Vertex angewandt wird.

Starten der Anwendung

Um die Applikation zu starten ist es notwendig ein **.env** File mit den notwendigen Parametern zu erstellen. Für die mitgelieferten Models stehen bereits vordefinierte .env Files zur Verfügung welche nach dem entsprechenden Model benannt sind (z.B. pompeii.env). Neben den von CHC++ benötigten Parametern müssen hier auch folgende Parameter für den Out-of-Core Algorithmus spezifiziert werden:

Parameterübersicht

Parameter	Typ	Beschreibung
doAreaCalculation	bool	Ob die projizierte Fläche der Bounding Box berechnet werden soll
areaCullThreshold	float	Minimale Fläche der proj. BB, damit überhaupt gerendert wird.
pixelDarkenFactor	float	Faktor, mit dem die Farbe multipliziert wird, um sie dunkler zu machen.
areaRenderThreshold	float	Maximale Fläche der proj. BB damit pixelgerendert wird
nrOfLoadingThreads	int	Anzahl der HD→Ram Threads
eraseFromRam	bool	Ob die Geometries aus dem Ram gelöscht werden nachdem sie in den VRAM kopiert worden sind
eraseFromRamByProbability	bool	Ob die Geometries aus dem Ram gelöscht werden, wenn ihre Sichtbarkeitswahrscheinlichkeit unter *ramProbabilityThreshold* fällt
vramProbabilityThreshold	float	Minimale Sichtbarkeitswahrscheinlichkeit für VRAM
vramUsageThreshold	int	VRAM Speicher, der verwendet werden soll (Bytes)
ramProbabilityThreshold	float	Minimale Sichtbarkeitswahrscheinlichkeit für RAM
ramUsageThreshold	int	Maximaler RAM Speicher, der verwendet werden soll (Bytes)
ramUsageTargetFactor	float	% von ramUsageThreshold die dauerhaft belegt bleiben dürfen, alles darüber wird möglichst bald wieder bereinigt
maxNrOfProxiesTo ProcessInOneFrame	int	Maximale Anzahl der Geometries, die in einem Frame verarbeitet werden (erstellen/löschen der VBOs)
maxNrOfProxies ToDeleteInOneFrame	int	Maximale Anzahl der Geometries, die in einem Frame gelöscht werden (lazy deleting), außer der VRAM ist knapp, dann gilt hier maxNrOfProxiesToProcessInOneFrame. Der VRAM Cleaner arbeitet überhaupt nur, falls die RAM→VRAM Queue leer, oder der VRAM voll ist.
probabilityScalingFactor	int	GeometryProxy::getProbability() liefert folgende Funktion: *max((-NrOfFramesNotRendered / probabilityScalingFactor) + 1, 0)*, *probabilityScalingFactor* ist also jene Anzahl von Frames, die eine Geometry nicht gerendert werden muss, damit ihre Wahrscheinlichkeit 0 erreicht.
renderVertexArrays	bool	Ob Geometries, die im RAM, aber nicht im VRAM liegen, mit Vertex Arrays gerendert werden sollen.

Durch Drag & Drop des gewünschten .env Files auf die mitgelieferte **start.bat** Datei wird die Anwendung mit dem im .env File spezifizierten Model und den gewünschten Parametern gestartet.

Interface

Die jeweiligen eingestellten Out-of-Core Parameter werden auch im HUD angezeigt, sowie der derzeitige RAM und VRAM Verbrauch. Ebenfalls angezeigt wird die Länge der entsprechenden Lade Queue.

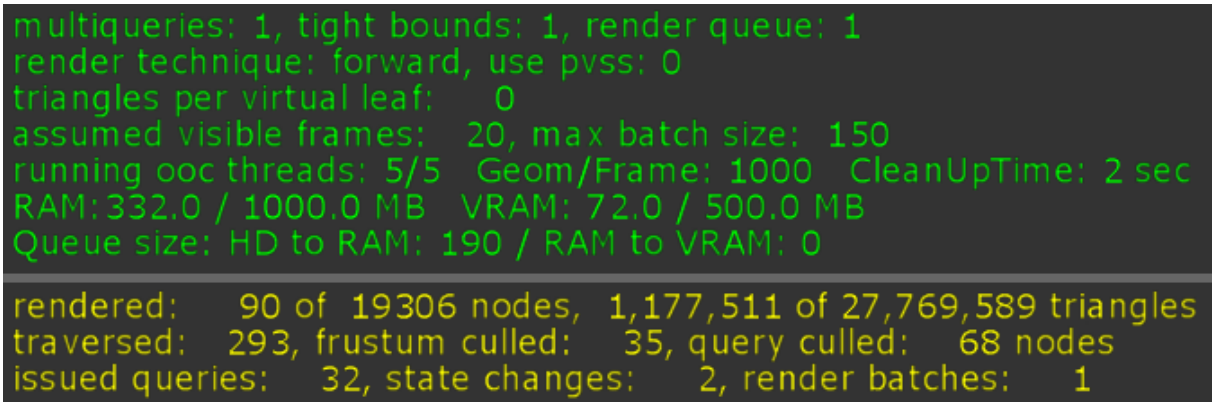


Abbildung 2: Screenshot des HUD

Mittels der folgenden Tasten ist es außerdem möglich einige Parameter während der Ausführung der Applikation zu verändern:

- **Page Up** - max. RAM um 50MB erhöhen
- **Page Down** - max. RAM um 50MB verringern
- **Pos1** - max. VRAM um 25MB erhöhen
- **Ende** - max. VRAM um 25MB verringern
- **Einfg** - *MaxNrOfProxiesToProcessInOneFrame* um 25 erhöhen
- **Entf** - *MaxNrOfProxiesToProcessInOneFrame* um 25 verringern

3. Implementierungsbeschreibung

Die Basis für dieses Projekt stellt das CHC++ Framework (1) dar, dessen wichtigste Elemente eine hierarchisch raumgliedernde Datenstruktur (Bounding Volume Hierarchie - BVH) und dazugehörige Traverser zum Aufbau einer Render-Queue sind. Der BVH Tree setzt sich aus Node Objekten zusammen, die gleichzeitig auch die Sichtbarkeitsinformationen kapseln. Dadurch wird ermöglicht, dass das unnötige Traversieren von ohnehin nicht sichtbaren Bereichen (Subtree) verhindert wird. Zur Evaluierung der Sichtbarkeit werden Occlusion-Queries in Kombination mit statistischen Werten (z.B. VisbilityPredictor Werte) verwendet.

Grundsätzlich wird der BVH Tree über SceneEntites konstruiert, die wiederum eine Menge an Shapes besitzen. Jede dieser Shapes stellt dabei eine Gruppe von den eigentlichen Geometry Objekten dar. Entscheidend für unsere Implementierung ist nun der Austausch dieser Geoemtry Objekte durch einen Proxy der den Zugriff auf die Daten nach einer bestimmten Strategie managt. Hierzu war es nötig sowohl einen eigenen Traverser (OutOfCoreTraverser, aufbauend auf dem CHCPlusPlusTraverser) zu implementieren, als auch, wie bereits in der Einleitung erwähnt, ein spezielles Modelformat zu entwerfen.

Modelformat demv2

Die Geometry Daten werden in eigene Files ausgelagert, sodass das .demv2 File nur als Indexfile fungiert und die eigentlichen Daten in anderen Files liegen. Im Index File stehen außer dem Dateinamen des Geometry Files ('.geom') und eines Offsets, da mehrere Geometries in einem File gebündelt werden können (um die riesige Anzahl von Files zu vermeiden, wenn jede Geometry in ein eigenes File geschrieben werden würde), noch die Bounding Box und der Vertexcount der jeweiligen GeometryProxy. *Details siehe Anhang.*

GeometryProxy

Der Zugriff auf die Geometrie wird mittels Proxy Objekten kontrolliert - Um die Changes in der Engine (dokumentiert unter (3)) möglichst gering zu halten, erbt der Proxy von Geometry.

Die Geometry kann nun unterschiedliche States *(GEOM_NOT_LOADED, GEOM_IN_RAM, GEOM_LOADING_IN_PROGRESS, GEOM_IN_VBO, GEOM_IN_RAM_AND_VBO)* annehmen die über den *OutOfCoreResourceManager (siehe folgende Punkte)* verändert werden. Der Proxy enthält zusätzlich eine Referenz auf das zugehörige (virtual) Leaf, die zur Berechnung einer Wahrscheinlichkeitsfunktion (GeometryProxy::getProbability()) dient.

Die verwendete Funktion ist

$$f(\text{nrOfFramesNotRendered}) = max\left(-\frac{\text{nrOfFramesNotRendered}}{\text{probabilityScalingFactor}} + 1, 0\right)$$

Hierbei entspricht der Parameter *probabilityScalingFactor* jener Anzahl von Frames bei denen einen Geometry nicht gerendert werden muss, damit ihre Wahrscheinlichkeit **0** erreicht.

Die Rendermethode berücksichtigt ebenfalls den aktuellen Ladestatus und rendert nur jene Daten, die sich im VRAM befinden. Optional können aber auch zu rendernde Objekte die sich jedoch nur im RAM befinden (z.B. wenn der VRAM voll ist, oder die VBOs aus Zeitmangel noch nicht erstellt werden konnten) mittels Vertex Arrays gerendert werden. Der Proxy bietet besonders hinsichtlich der Situation eines vollen VRAMs die Möglichkeit die Daten aus dem VRAM in den RAM zurück zu kopieren *(GeometryProxy::copyGeometryFromVBOtoRAM())* bzw. VBOs zu clearen *(GeometryProxy::unloadGeometryVBO)*. Analog hierzu gibt *GeometryProxy::unloadGeometryRAM* den RAM Speicher frei.

OutOfCoreTraverser

Ist eine Erweiterung des *CHC++ Traversers*, welcher beim Durchlaufen der Renderqueue die States der *GeometryProxies* berücksichtigt. Weiters enthält und managt dieser Traverser das im folgenden Punkt beschriebene *Pixelrendern*.

Projected-Area Culling

Die projizierte Fläche (4) der Node-Bounding-Box wird mittels Konturintegral berechnet (inklusive Sutherland-Hodgman Clipping).

Fällt die projizierte Fläche unter einen gewissen Schwellwert (per .env einstellbar) wird dieser Node komplett gecullt und die Traversierung gestoppt.

Liegt jedoch die projizierte Fläche zwischen diesem und einem weiteren (größeren) Schwellwert (ebenfalls einstellbar) wird die Position und Farbe des betrachteten Objekts in eine Queue eingereiht.

Nachdem alle nicht gecullten Objekte gezeichnet wurden, wird diese Queue abgearbeitet und statt Triangles wird für den jeweiligen Node ein Pixelrechteck (glDrawPixels) im Zentrum der BoundingBox gerendert (austauschbar in *renderPoints*).

Dies erlaubt uns die Traversierung schon auf einer der oberen Ebenen abzubrechen falls die Fläche eines Nodes schon zu gering ist um einen visuellen Einfluss zu haben.

OutOfCoreResourceManager

Ein eigener Resource Manager wurde implementiert um die States der Proxy Objekte zu verwalten. Der Manager besitzt zwei Queues (HDD→RAM und RAM→VRAM), wobei die HDD→RAM Queue von beliebig vielen Threads abgearbeitet wird. Die RAM→VRAM Queue kann nicht in Threads abgearbeitet werden, da zum Erstellen der VBO's die OpenGL States konsistent bleiben müssen. Um die Verwaltung der Threads und die Synchronisation zu erleichtern wurde GLFW eingebunden, da wir mit der Verwendung dieser Threading-API aus CG2 vertraut waren. Zusätzlich weiß der Ressource Manager zu jeder Zeit welche Proxies in RAM bzw. VRAM liegen, und wie viel Speicher belegt ist, und kann jederzeit Ressourcen freigeben. Gelöscht werden jene Proxies, die die geringste Sichtbarkeitswahrscheinlichkeit *(GeometryProxy::getProbability())* haben.

Wie schon allgemein beschrieben, läuft das Laden einer Geometrie in zwei Stufen ab. Zuerst wird der jeweilige Proxy in die *OutOfCoreResourceManager::load(GeometryProxy* proxy)* Methode geschickt. Diese Methode entscheidet anhand des States des Proxies, was mit diesem geschieht - wenn er nicht geladen ist, wird er in die HD→RAM Queue eingereiht, welche von mehreren Threads parallel abgearbeitet wird. Falls der Proxy bereits im RAM, aber nicht im VRAM ist, wird geprüft ob er in den VRAM geladen werden soll *(GeometryProxy::predictor())*; Falls dieser Test erfolgreich ist, wird er in die RAM→VRAM Queue eingereiht, die einmal pro Frame abgearbeitet wird. Ist der Proxy bereits im VRAM, tut der Manager nichts.

Wenn einer der HD→RAM Threads einen Proxy erfolgreich in den RAM geladen hat, prüft er wieder mit *GeometryProxy::predictor(),* ob der Proxy in den VRAM geladen werden soll, und reiht ihn in die RAM→VRAM Queue ein, falls dieser Test erfolgreich ist.

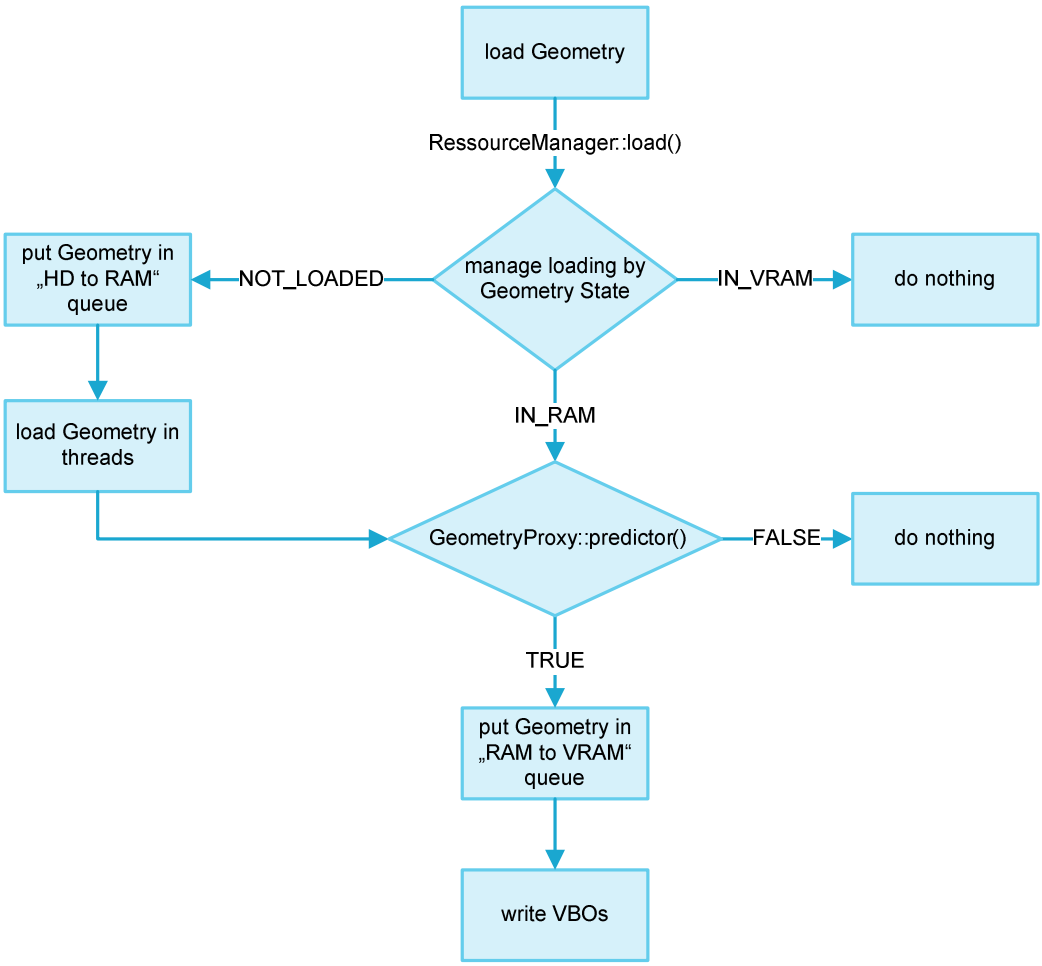


Abbildung 3: Ablaufdiagramm des Ladens einer Geometry

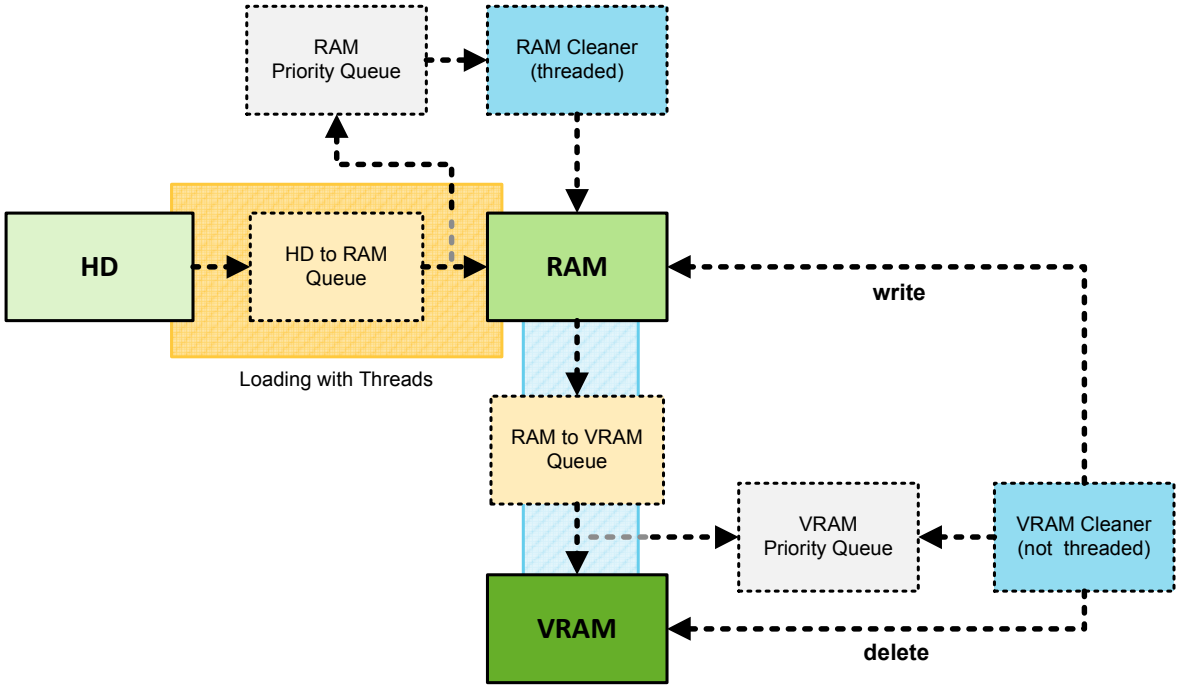


Abbildung 4: Schematische Darstellung des OutOfCoreResourceManagers

Wie oben erwähnt hält der Resource Manager eine Liste von allen Proxies, die zur Zeit im RAM bzw. VRAM sind. Diese Liste ist als *PrioritySortableGeometryProxyQueue* implementiert, das heißt ihre Einträge sind (mit *GeometryProxy::getProbability()* als Key) absteigend sortierbar, und werden genau einmal pro Frame sortiert. Diese Sortierung ist für das Freigeben von Speicher wichtig, weil so die Liste einfach von hinten nach vorne abgearbeitet werden kann, bis die Wahrscheinlichkeit eines Proxies über einen Schwellwert steigt.

Es existieren zwei Mechanismen, die das Aufräumen der beiden Speicher übernehmen:

- **RAM-Cleaner Thread:** Dieser Thread löscht Geometrie aus dem RAM, je nach den gewählten Settings nach Wahrscheinlichkeit oder nur wenn der verfügbare Speicherplatz knapp wird.
- **VRAM-Cleaner:** Diese Methode wird einmal pro Frame gecallt (unmittelbar bevor RAM→VRAM Queue abgearbeitet wird). Hier wird Geometrie aus dem VRAM gelesen und in den RAM geschrieben, falls sie nicht ohnehin schon im RAM existiert. Wie viele VBOs hier abgearbeitet werden entscheidet sich nach folgender Heuristik: Falls die RAM→VRAM Queue derzeit leer ist, werden maximal *maxNrOfProxiesToDeleteInOneFrame* VBOs gelöscht. Falls die RAM→VRAM Queue nicht leer ist, wird grundsätzlich nichts gelöscht um Zeit zu sparen (Laden ist wichtiger als Löschen), außer der aktuelle VRAM Verbrauch ist nahe an der Grenze (bei 90%), in diesem Fall werden ebenso viele VBOs gelöscht, wie Einträge in der RAM→VRAM Queue vorhanden sind, maximal jedoch *maxNrOfProxiesToProcessInOneFrame* (unter der Annahme dass alle VBOs ca. gleich groß sind, sollte das genug Speicher freigeben, aber nicht unnötig lange blockieren).

4. Versuche

Die folgenden Benchmarks wurden mit einem aufgezeichneten Pfad erstellt, um dieselben Bedingungen reproduzierbar zu machen und somit ein objektives Ergebnis zu erhalten.

Das TimeDiff Diagramm zeigt die Differenz in den Framezeiten zweier Walkthroughs, bei einem positiven Ausschlag ist also v1 besser, bei einem negativen v2.

SSD vs HDD

Um zu überprüfen, wie stark das Laden von der Festplatte die Performance beeinträchtigt, haben wir probeweise das Model von einer SSD geladen. Eine Performancesteigerung war damit nicht erreichbar, allerdings fällt auf, dass beim Starten die Szene schneller aufgebaut wird.

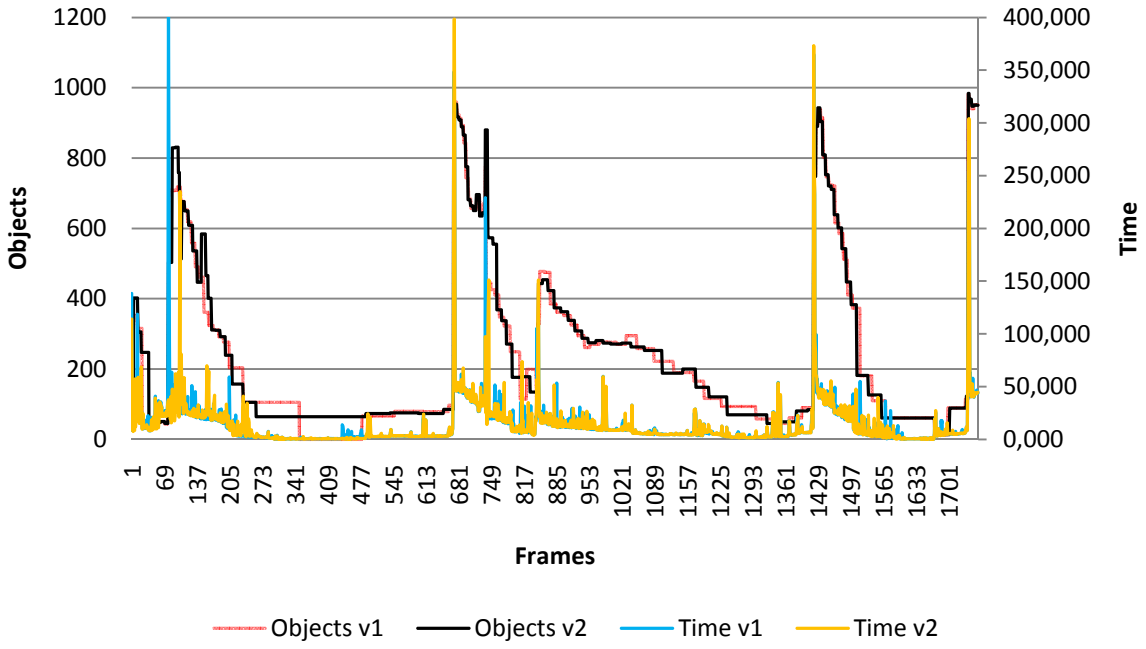
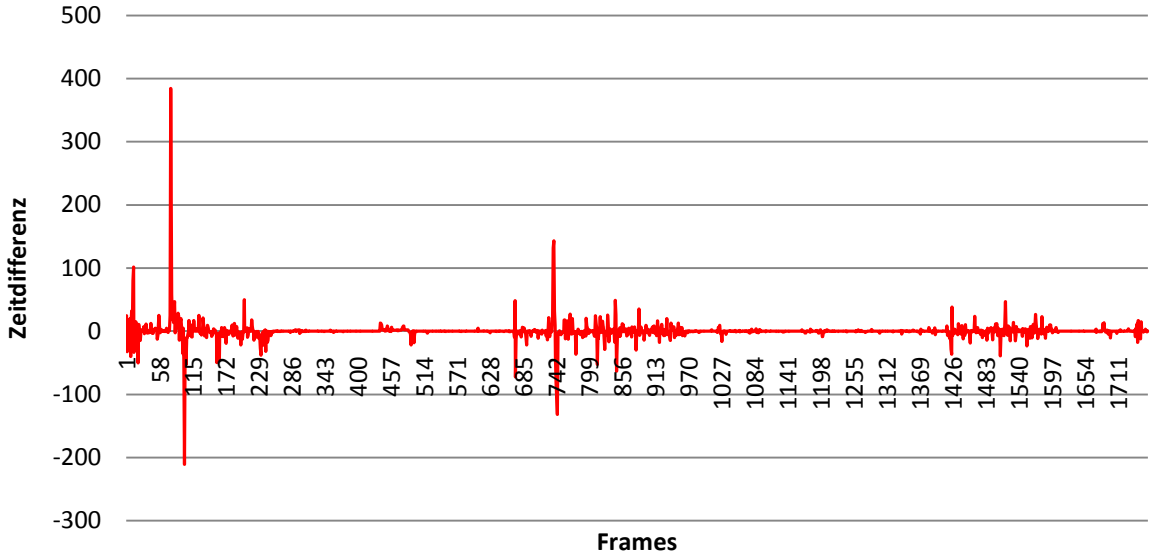


Abbildung 5 - v1 - SSD und v2 - HDD



Tighter Bounds

Auf Olivers Rat hin haben wir getestet ob das Ein-/Ausschalten der "Tighter Bounds" Verbesserungen bringt. Auch hier war nur eine minimale Verbesserung zu erkennen, bei eingeschalteten Tighter Bounds war die durchschnittliche Zeit / Frame war um eine Millisekunde besser, die maximale Zeit / Frame um 14ms.

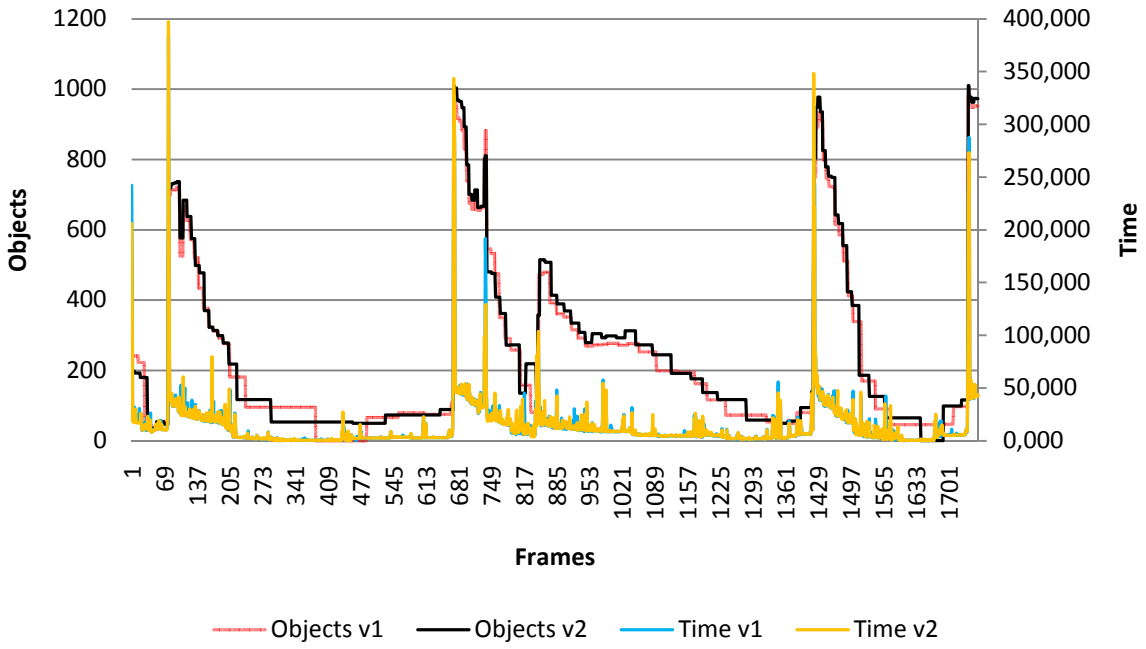
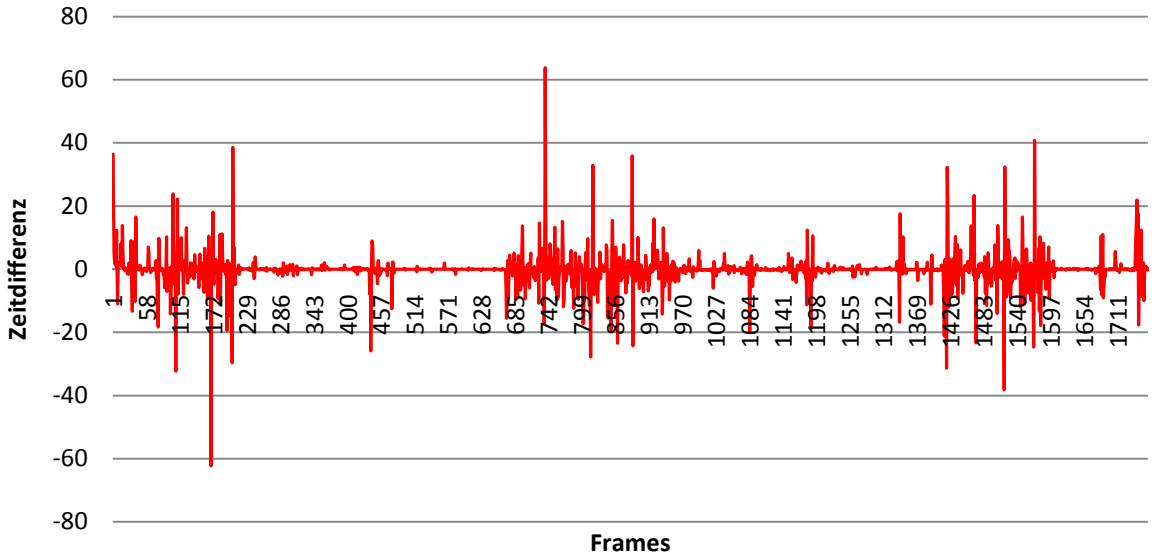


Abbildung 6 - v1 - Tighter Bounds 10 und v2 - ohne Tighter Bounds



CHC++ vs OOC

Vergleich zwischen CHC++ (Engine ohne Out of Core) und unserem Out of Core Algorithmus. Anzumerken ist das beim CHC++ das komplette Model (Powerplant) in den VRAM geladen wird wodurch das Starten der Applikation etwas länger dauert. Bei unserem Out of Core Algorithmus werden die Teile des Models hingegen nach und nach sichtbar. Die ab und zu auftretenden positiven Spikes im TimeDiff Diagramm sind hier Stellen wo Geometrie nachgeladen werden muss. Weiters ist anzumerken das bei CHC++ auch Skybox etc. gerendert werden, welche in unserer Implementierung weggelassen wurden.

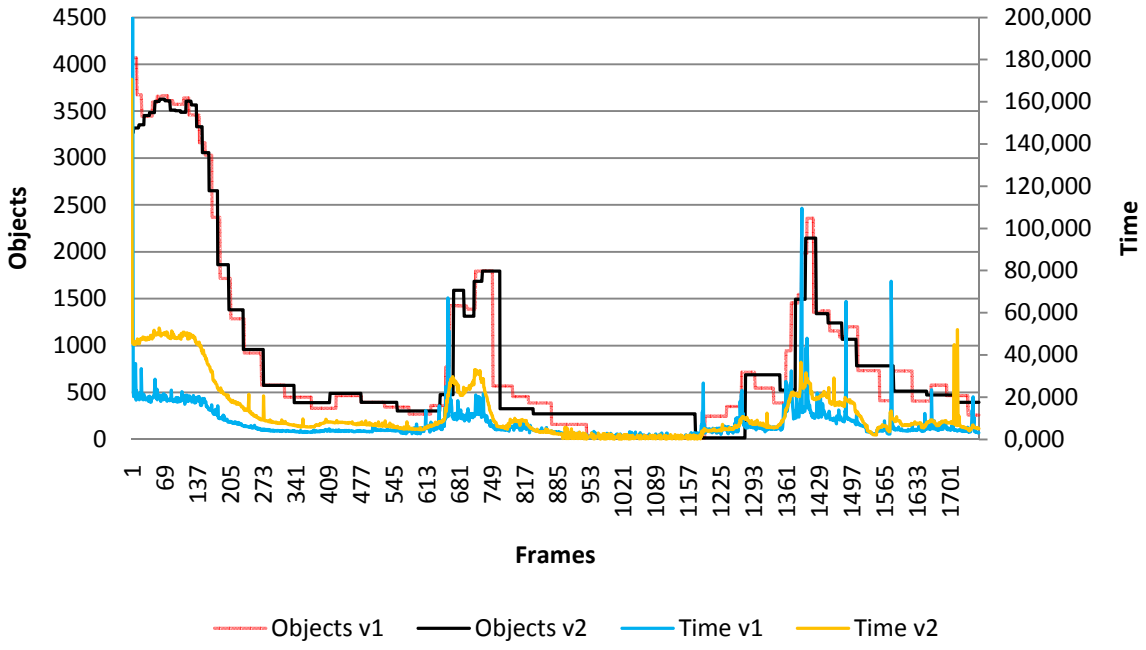
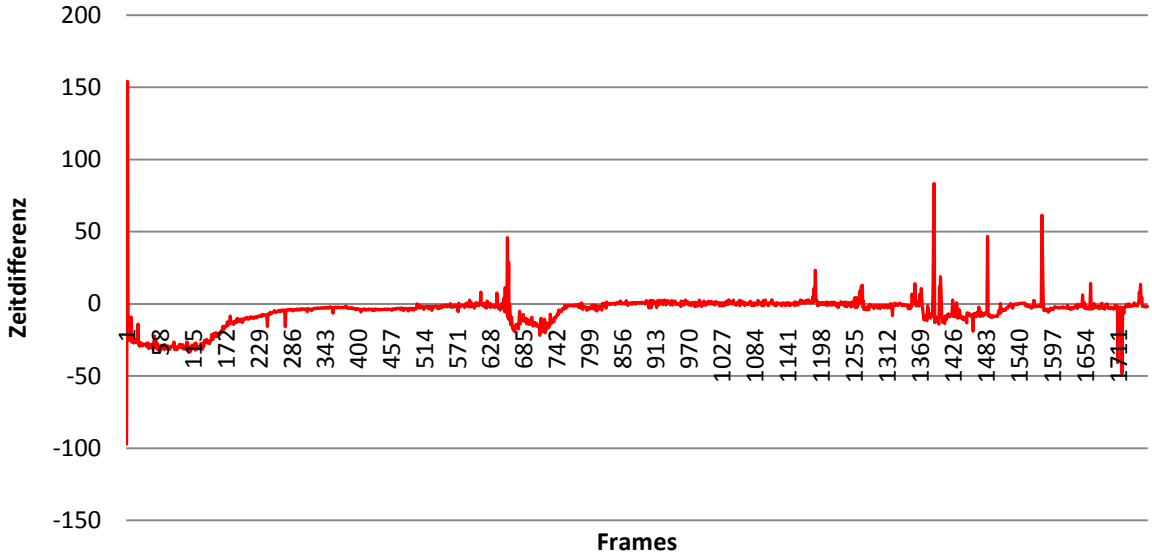


Abbildung 7 - v1 - OOC und v2 - CHC++



5. Anhang

Details zum Modelformat

*.demv2

```
> # ------------------------------------------------------------------------
> # Header
> # ------------------------------------------------------------------------
>
> DEMv2PiN                                  char[8]                 #Identification
> version                                   int                     #Versionsnummer

  # ------------------------------------------------------------------------
  # Textures
  # ------------------------------------------------------------------------

  TextureCount                              int
  FOR (TextureCount)                                                #Textures
  {
          texnameSize                       int
          texName                           char[texnameSize]       #relativer Pfad zum Texturfile ausgehend vom
                                                                    #dem Folder wo das .demv2 file liegt.
          boundS                            int
          boundT                            int
  }

  # ------------------------------------------------------------------------
  # Shapes
  # ------------------------------------------------------------------------

  numShapes                                 int
  FOR (numShapes)                                                   #Shapes
  {
>         geomFilenameSize                  int
>         geomFilename                      char[geomFilenameSize]  #relativer Pfad zum Geomfile (.geom) ausgehend
                                                                    #vom dem Folder wo das .demv2 file liegt.
>         geomFileOffset                    int                     #offset im GeomFile (in komprimierten Bytes)
>         bBoxMin                           Vector3                 #bounding box min
>         bBoxMax                           Vector3                 #bounding box max
>         vertexCount                       int                     #nr of vertices in this geometry

          # ------------------------------------------------------------------------
          # Material
          # ------------------------------------------------------------------------

          texID                             int                     #-1 if no texture exists
          alphaTestEnabled                  bool
          cullFaceEnabled                   bool
          hasGLmaterial                     bool
          IF (hasGLmaterial)
          {
                    ambient                 Vector3
                    diffuse                 Vector3
                    emmissive               Vector3
                    specular                Vector3
          }
  }

  # ------------------------------------------------------------------------
  # Scene Entities
  # ------------------------------------------------------------------------

  entityCount                               int
  FOR (entityCount)
  {
          hasTrafo                          bool
          IF (hasTrafo)
          {
                    trafo                   Matrix4x4
          }
          numLODs                           int
          FOR (numLODs)                                             #LODs must be ordered from smallest distance
                                                                    #to largest
          {
                    dist                    float
                    numShapes               int
                    FOR (numShapes)
                    {
                              shapeID       int
                    }
          }
  }
```

*.geom

```
  # ------------------------------------------------------------------------
  # Geometry
  # ------------------------------------------------------------------------

> # removed vertexCount (now in .demv2 file, not in .geom file)
  FOR vertexCount                                                   #Vertices
  {
          Vertex                                Vector3
  }
  FOR (vertexCount)                                                 #Normals
  {
          Normal                                Vector3
  }
> tangentCount                                  int                 #Can only be 0 or equal to vertexCount,
                                                                    #depending if we have tangents or not
> FOR (tangentCount)                                                #Tangents
> {
>         Tangent                               Vector3
> }
  texCoordCount                                 int                 #Can only be 0 or equal to vertexCount,
                                                                    #depending if we have texcoords or not
  FOR (texCoordCount)                                               #Texcoords
  {
          TexCoord                              Texcoord2
  }
```

6. Literaturverzeichnis

1. **Oliver Mattausch, Jiří Bittner, Michael Wimmer.** CHC++: Coherent Hierarchical Culling Revisited. *Computer Graphics Forum (Proceedings Eurographics 2008), 27(2):221-230, April 2008.* 2008.

2. Cg Toolkit. [Online] http://developer.nvidia.com/object/cg_toolkit.html.

3. Trac - OutOfCore Projekt. [Online] http://poseidon.shacknet.nu:8000/OutOfCore.

4. **Tobler, Dieter Schmalstieg and Robert F.** Fast Projected Area Computation for Three-Dimensional Bounding Boxes. *journal of graphics, gpu, and game tools.* 1999.

5. **Beat Brüderlin, Mathias Heyer and Sebastian Pfützner.** Interviews3D:A Platform forInteractiveHandling of MassiveData Sets. *IEEE Computer Graphics and Applications.* 2007.