Dynamic loading of 3D models

Computing Project - PROJ-H-402

Tim Lenertz

ULB, MA1 INFO

May 19, 2014

Contents

Introduction

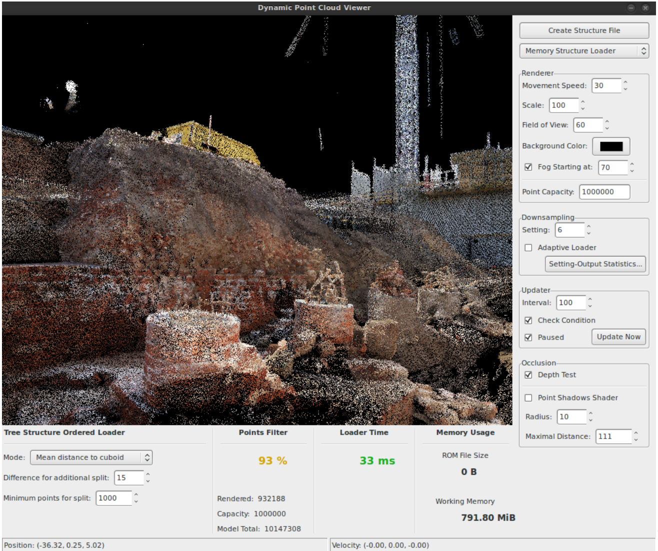
- Large XYZRGB point clouds
- Composite point clouds
- High density 3D scanner
- $\Rightarrow$ Too many points to process as whole
- **Goal**: Dynamically extract smaller sets for visualization
- Filtering techniques
- Data structure and file format

Example

Unfiltered render
$\Rightarrow$ buffer overflow

Using Octree structure
8 downsampled LOD regions

Application Screenshot



Mechanism

- Extract part P' of model P
- Relative to current camera position, orientation, etc.
- P' filtered to remove unnecessary parts
- $\rightarrow$ Fit into renderer capacity C
- $\rightarrow$ Retain visual quality
- Implemented using underlying *data structure*
- Data structure created in preprocessing step
- Extraction of P' is time critical

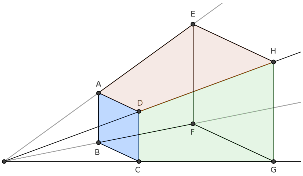
Techniques

- Theoretical description of f_P
- Without regard for data structure
- Techniques used:

  View frustum culling Render only visible parts of model
  Downsampling Reduce point density in distant regions
  Occlusion culling Don't render hidden surfaces

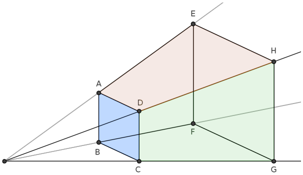
Frustum culling

- Remove invisible points
- Most effective part
- Often removes $\geq 1/2$
- With data structure: Remove entire regions
- $\rightarrow$ Before sending to GPU



Frustum

- Frustum = 6 planes
  (near, far, top, bottom, left, right)
- Apex = camera
- Entirely defined by 4x4 projection matrix $\boldsymbol{M}$
- $\boldsymbol{M} = \boldsymbol{P} \times \boldsymbol{V}$
- $\boldsymbol{V}$ = view matrix
  camera position and orientation
- $\boldsymbol{P}$ = projection matrix
  perspective or parallel 3D to 2D projection



Projection matrix

- Using *homogeneous coordinates*
- For 4x4 matrices
- Vector $[x, y, z, w] \rightarrow [\frac{x}{w}, \frac{y}{w}, \frac{z}{w}]$
- $w = 0$: direction $\qquad w = 1$: position
- Non-linear: Division by w
- Allows for perspective foreshortening
- Perspective projection matrix:

$$\boldsymbol{P} = \begin{bmatrix} \frac{f}{w/h} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{z_{\text{far}}+z_{\text{near}}}{z_{\text{near}}-z_{\text{far}}} & \frac{2\times z_{\text{far}}\times z_{\text{near}}}{z_{\text{near}}-z_{\text{far}}} \\ 0 & 0 & -1 & 0 \end{bmatrix} \text{ with } f = \frac{1}{\tan(\frac{\lambda}{2})}$$

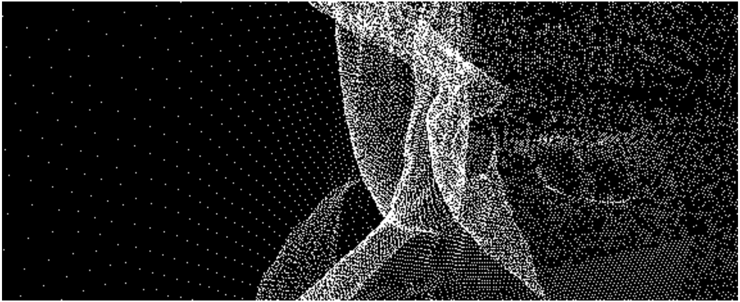
Downsampling

- Reduce point density
- Foreshortening $\rightarrow$ downsample distant parts
- Points are always located on (unknown) surfaces
- Density $= n/A$, locally constant, but can vary in composite models
- Downsampling ratio: $r(d) \in [0, 1]$
- d = distance to camera (of data structure region)
- Variants:
  - Weighted points
  - LOD (level of detail) regions

Weighted points

- Assign weight $w \in [0, 1]$ to each point
- Downsampled points = subset
  $P' = \{p \in P : w(p) < r(d(p))\}$
- w need to be uniformly distributed among points
- $\rightarrow$ Associate random weights
- $r(d)$ can take any value
- $\rightarrow$ Visual continuity
- Random $\rightarrow$ Irregular pattern

Weighted points (2)



Random weights $\Rightarrow$ irregular pattern

LOD regions

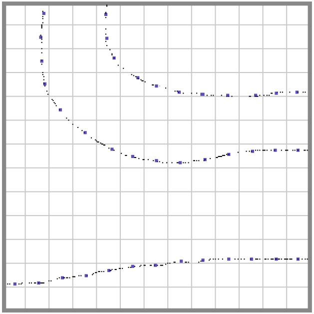
- Precompute downsampled point sets
- Usually 4, 8 or 16
- Fixed r
- No time constraint for downsampling
- Visual discontinuitites

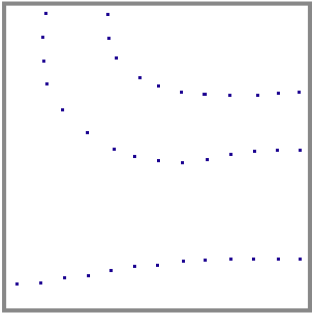
LOD regions (2)

Uniform downsampling

LOD regions

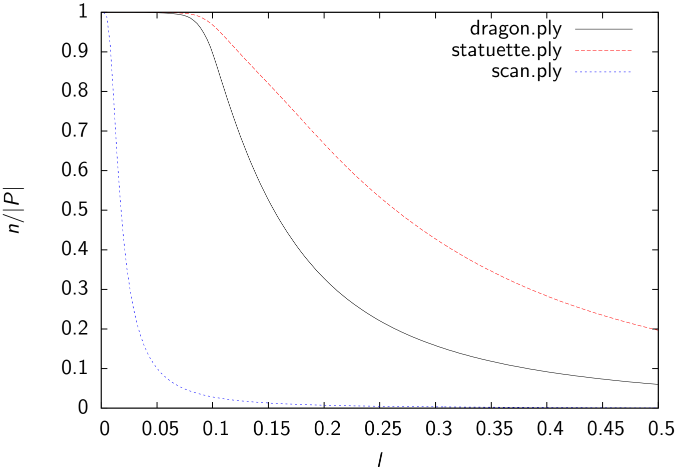
- Algorithm to get more regular pattern
- Constant r
- Cubes grid with side length l
- Take only mean point
- Find l for expected r: Dichotomic search





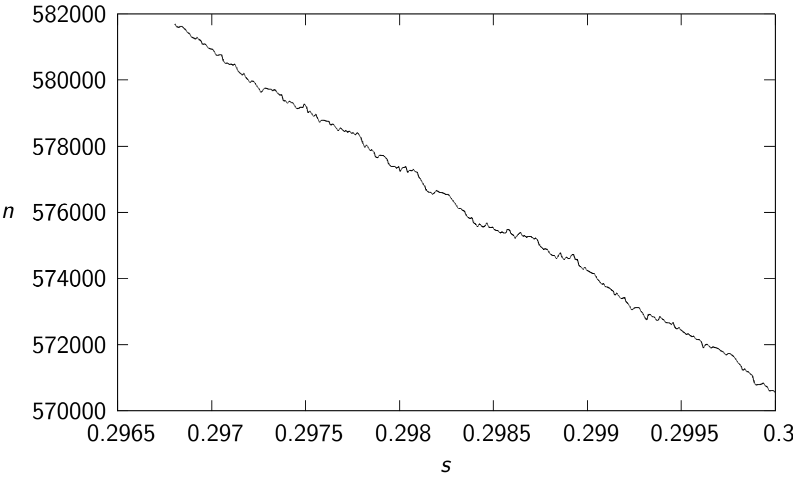
LOD regions (2)

Cubes side length l VS output number of points n



LOD regions (3)

Not fully monotonic. Close-up:



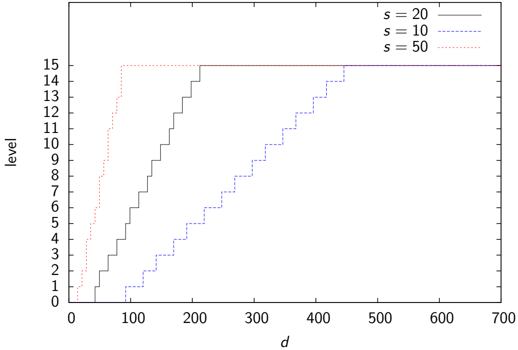
Downsampling ratio choice

- Definition of $r(d)$
- Decreasing with d
- Distance $\rightarrow$ Level $\rightarrow$ Ratio
- For *LOD regions*: L possible levels
- For *weighted points*: Level can be continuous

Downsampling level

Downsampling level $\ell(d, s)$:

$$\ell(d,s) = \begin{cases} \min\{\frac{d-d_0}{\Delta d}, L-1\} & d > d_0 \\ 0 & d \leq d_0 \vee s = 0 \end{cases}$$

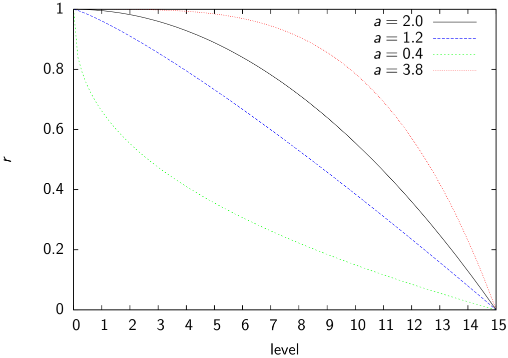


L = nb of levels
s = setting
$b = \frac{250}{s}$
$\Delta d = b$
$d_0 = b^{1.3}$

Downsampling ratio

Downsampling ratio $r(\ell)$ for level ℓ:

$$r_\ell(\ell, a, L, n_{\text{total}}, n_{\text{min}}) = 1 - (1 - r_{\text{min}}) \left(\frac{\ell}{L-1}\right)^a$$

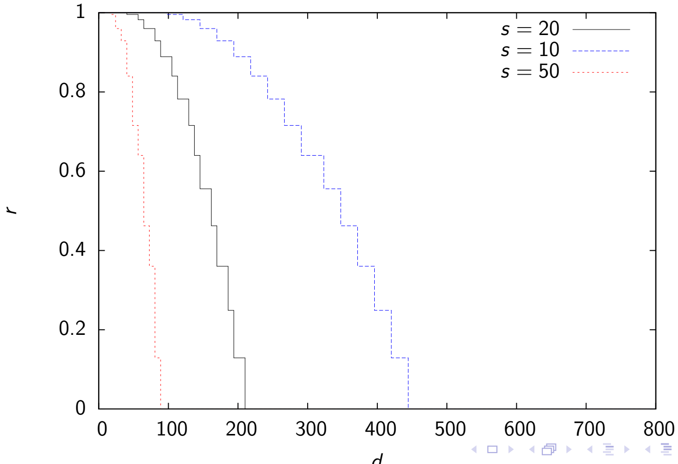


a = amount (const)
n_{total} = total points
n_{min} = minimal pts
$r_{min} = \max\{\frac{n_{\text{min}}}{n_{\text{total}}}, 1\}$

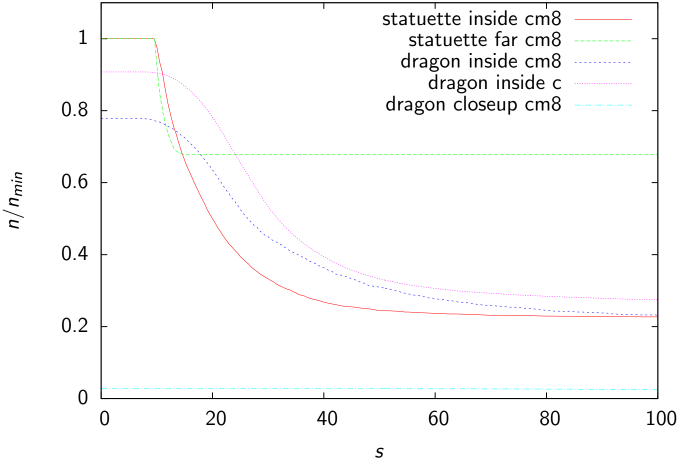
$a = 2$ good choice

Downsampling ratio for distance

$$r(d) = r_\ell \circ \ell$$



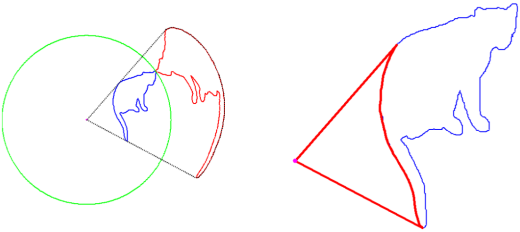
Effect of downsampling setting s



Occlusion culling

- = Hidden surface removal
- Surfaces unknown (only points)
- Several approximative algorithms

HPR operator



1. Center points around camera
2. Spherical flipping transformation:
   $p' = p + 2(R - |p|)\frac{p}{|p|}$
3. Compute convex hull (slow)
4. Retain original points whose image is in c.h.

Data structures

- Way to organize points
- Serialized into 1D memory
- $\rightarrow$ Using *HDF5* file format
- Filtering implemented on top of data structure
- Prefer reading few long segments
- $\rightarrow$ Try to keep close points together in serialization

Cuboid regions

- Technique: Subdivide into cuboid regions
- Downsampling ratio $r(d)$ per region
- d: *point-to-cuboid* distance to camera
- $\rightarrow$ Need to choose compromise definition

Cubes structure

- Use grid of cubic regions
- Common side length
- Extract points from cubes that are inside frustum
- $\rightarrow$ Cuboid-frustum intersection test
- Both *weighted points* and *LOD regions* variants

Cubes structure (2)

TableView - cubes - / - /home/tl/Documents/ULB/...

Table

1816, x ... 13

	x	z	y	data_start	data_length
1800	12	5	5	3050671	1673
1801	12	5	6	3052344	1357
1802	12	6	1	3053701	1699
1803	12	6	2	3055400	2914
1804	12	6	3	3058314	3305
1805	12	6	4	3061619	3558
1806	12	6	5	3065177	3217
1807	12	6	6	3068394	17
1808	12	7	1	3068411	697
1809	12	7	3	3069108	3
1810	12	7	4	3069111	129
1811	12	7	5	3069240	190
1812	12	8	1	3069430	4241
1813	12	8	5	3073671	2641
1814	12	9	1	3076312	716
1815	12	9	5	3077028	97
1816	13	-8	10	3077125	23
1817	13	-8	11	3077148	13
1818	13	-8	13	3077161	697
1819	13	-7	10	3077858	1082
1820	13	-7	11	3078940	644
1821	13	-7	12	3079584	272
1822	13	-2	3	3079856	137
1823	13	-2	4	3079993	379
1824	13	-2	5	3080372	10
1825	13	-1	2	3080382	1162
1826	13	-1	3	3081544	2562
1827	13	-1	4	3084106	2684
1828	13	-1	5	3086790	2155
1829	13	0	0	3088945	18
1830	13	0	1	3088963	2017

TableView - points - / - /home/tl/Documents/ULB/INFO4 2013-2014/...

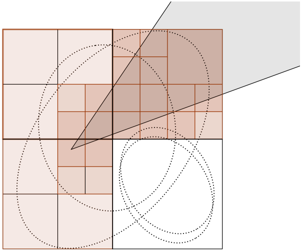
Table

3077125... 65.035164

	x	y	z	r	g	b	weight
3077119	61.994484	45.045628	27.373087	255	255	255	0.064114...
3077120	62.012463	45.0	28.257986	255	255	255	0.045634...
3077121	61.890373	45.244785	27.857601	255	255	255	0.020706...
3077122	62.229256	45.039577	27.506832	255	255	255	0.020490...
3077123	62.002777	45.120197	28.127573	255	255	255	0.014154...
3077124	62.20014	45.101017	28.17547	255	255	255	6.07532E-4
3077125	65.035164	-35.075504	51.281597	255	255	255	0.9096286
3077126	65.154396	-35.027958	51.05727	255	255	255	0.897825...
3077127	65.020164	-35.07613	51.38459	255	255	255	0.889138...
3077128	65.11555	-35.01403	51.724743	255	255	255	0.8535559
3077129	65.10079	-35.052265	51.190437	255	255	255	0.810554...
3077130	65.12102	-35.024574	51.669243	255	255	255	0.8084886
3077131	65.058174	-35.057884	51.00906	255	255	255	0.7894411
3077132	65.20942	-35.013012	51.22606	255	255	255	0.6766508
3077133	65.03881	-35.075005	51.096527	255	255	255	0.6345862
3077134	65.18239	-35.01557	51.472485	255	255	255	0.519672...
3077135	65.11947	-35.043736	51.315918	255	255	255	0.389836...
3077136	65.15309	-35.023853	51.53493	255	255	255	0.3256594
3077137	65.01788	-35.034615	51.78424	255	255	255	0.299494...
3077138	65.183655	-35.019012	51.366962	255	255	255	0.2085842
3077139	65.0538	-35.049717	51.651546	255	255	255	0.200643...
3077140	65.16879	-35.0142	51.615692	255	255	255	0.1898439
3077141	65.166115	-35.012215	51.64302	255	255	255	0.171629...
3077142	65.16621	-35.005024	51.68238	255	255	255	0.168116...
3077143	65.02638	-35.029736	50.847965	255	255	255	0.102112...
3077144	65.037384	-35.04551	51.712536	255	255	255	0.091147...
3077145	65.076	-35.04837	51.561733	255	255	255	0.039059...
3077146	65.095505	-35.0281	50.941357	255	255	255	0.012541...
3077147	65.1004	-35.04525	51.44652	255	255	255	0.011265...
3077148	65.07599	-35.050354	59.488186	255	255	255	0.9128186
3077149	65.02184	-35.08368	59.499313	255	255	255	0.7466456

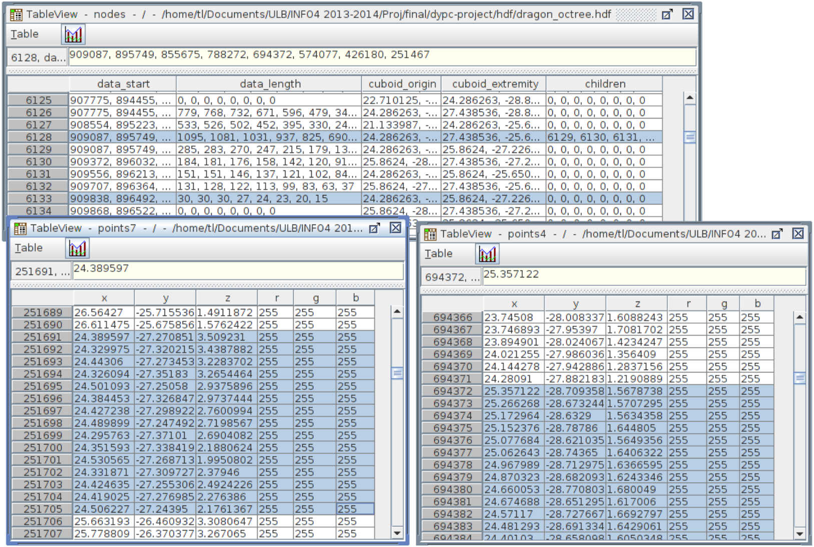
Tree structures

- Recursively subdivide
- ...until $\leq$ *leaf capacity* points in cuboid
- Points in any node remain in one segment
- Can include/exclude large nodes
- $\rightarrow$ No need to test all leaves
- Variants: Octree, KdTree



Octree Example

Tree structures (2)



TableView - nodes - / - /home/tl/Documents/ULB/INFO4 2013-2014/Proj/final/dypc-project/hdf/dragon_octree.hdf

Table

6128, da... | 909087, 895749, 855675, 788272, 694372, 574077, 426180, 251467

	data_start	data_length	cuboid_origin	cuboid_extremity	children
6125	907775, 894455, ...	0, 0, 0, 0, 0, 0, 0, 0	22.710125, -...	24.286263, -28.8...	0, 0, 0, 0, 0, 0, 0, 0
6126	907775, 894455, ...	779, 768, 732, 671, 596, 479, 34...	24.286263, -...	27.438536, -28.8...	0, 0, 0, 0, 0, 0, 0, 0
6127	908554, 895223, ...	533, 526, 502, 452, 395, 330, 24...	21.133987, -...	24.286263, -25.6...	0, 0, 0, 0, 0, 0, 0, 0
6128	909087, 895749, ...	1095, 1081, 1031, 937, 825, 690...	24.286263, -...	27.438536, -25.6...	6129, 6130, 6131, ...
6129	909087, 895749, ...	285, 283, 270, 247, 215, 179, 13...	24.286263, -...	25.8624, -27.226...	0, 0, 0, 0, 0, 0, 0, 0
6130	909372, 896032, ...	184, 181, 176, 158, 142, 120, 91...	25.8624, -28...	27.438536, -27.2...	0, 0, 0, 0, 0, 0, 0, 0
6131	909556, 896213, ...	151, 151, 146, 137, 121, 102, 84...	24.286263, -...	25.8624, -25.650...	0, 0, 0, 0, 0, 0, 0, 0
6132	909707, 896364, ...	131, 128, 122, 113, 99, 83, 63, 37	25.8624, -27...	27.438536, -25.6...	0, 0, 0, 0, 0, 0, 0, 0
6133	909838, 896492, ...	30, 30, 30, 27, 24, 23, 20, 15	24.286263, -...	25.8624, -27.226...	0, 0, 0, 0, 0, 0, 0, 0
6134	909868, 896522, ...	0, 0, 0, 0, 0, 0, 0, 0	25.8624, -28...	27.438536, -27.2...	0, 0, 0, 0, 0, 0, 0, 0

TableView - points7 - / - /home/tl/Documents/ULB/INFO4 201...

Table

251691, ... | 24.389597

	x	y	z	r	g	b
251689	26.56427	-25.715536	1.4911872	255	255	255
251690	26.611475	-25.675856	1.5762422	255	255	255
251691	24.389597	-27.270851	3.509231	255	255	255
251692	24.329975	-27.320215	3.4387882	255	255	255
251693	24.44306	-27.273453	3.2283702	255	255	255
251694	24.326094	-27.35183	3.2654464	255	255	255
251695	24.501093	-27.25058	2.9375896	255	255	255
251696	24.384453	-27.326847	2.9737444	255	255	255
251697	24.427238	-27.298922	2.7600994	255	255	255
251698	24.489899	-27.247492	2.7198567	255	255	255
251699	24.295763	-27.37101	2.6904082	255	255	255
251700	24.351593	-27.338419	2.1880624	255	255	255
251701	24.530565	-27.268713	1.9950802	255	255	255
251702	24.331871	-27.309727	2.37946	255	255	255
251703	24.424635	-27.255306	2.4924226	255	255	255
251704	24.419025	-27.276985	2.276386	255	255	255
251705	24.506227	-27.24395	2.1761367	255	255	255
251706	25.663193	-26.460932	3.3080647	255	255	255
251707	25.778809	-26.370377	3.267065	255	255	255

TableView - points4 - / - /home/tl/Documents/ULB/INFO4 20...

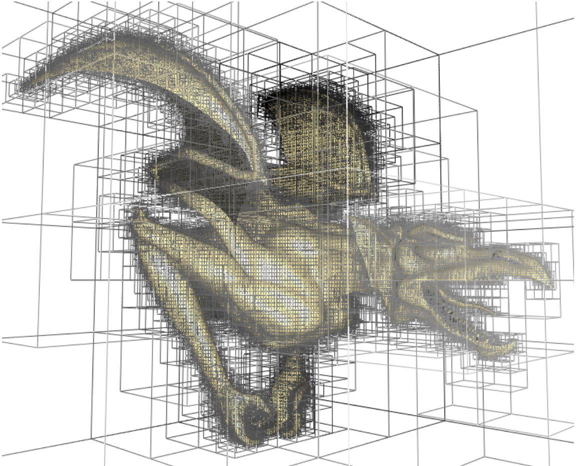
Table

694372, ... | 25.357122

	x	y	z	r	g	b
694366	23.74508	-28.008337	1.6088243	255	255	255
694367	23.746893	-27.95397	1.7081702	255	255	255
694368	23.894901	-28.024067	1.4234247	255	255	255
694369	24.021255	-27.986036	1.356409	255	255	255
694370	24.144278	-27.942886	1.2837156	255	255	255
694371	24.28091	-27.882183	1.2190889	255	255	255
694372	25.357122	-28.709358	1.5678738	255	255	255
694373	25.266268	-28.673244	1.5707295	255	255	255
694374	25.172964	-28.6329	1.5634358	255	255	255
694375	25.152376	-28.78786	1.644805	255	255	255
694376	25.077684	-28.621035	1.5649356	255	255	255
694377	25.062643	-28.74365	1.6406322	255	255	255
694378	24.967989	-28.712975	1.6366595	255	255	255
694379	24.870323	-28.682093	1.6243346	255	255	255
694380	24.660053	-28.770803	1.680049	255	255	255
694381	24.674688	-28.651295	1.617006	255	255	255
694382	24.57117	-28.727667	1.6692797	255	255	255
694383	24.481293	-28.691334	1.6429061	255	255	255

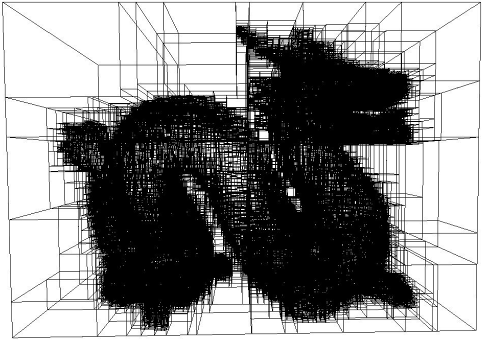
Octree

- Regions always cubes
- Split into 8 child nodes



Octree

- Split into 2 child nodes
- Split plane = median
- $\Rightarrow$ balanced tree
- in X, Y, Z, X... axis (depth modulo 3)



Piecewise construction

- One *Outer tree* without points
- Several *inner trees* at leaves of outer tree
- Different tree types possible
- $\rightarrow$ KdTree-Median good for outer tree
- Load one+ inner tree at a time
- $\rightarrow$ Limit memory usage
- $\rightarrow$ Parallization possible

Implementation

libdypc.so Library containing all algorithms / structures

viewer GUI front-end and dynamic renderer

Implemented in C++
Pure C library API interface

Streaming Mechanism

- 2 threads
  - **renderer** main GUI, renders from point buffer
  - **loader** periodically extracts new point set
- Update via OpenGL buffer swap
- $\rightarrow$ Loader writes directly into GPU buffer
- Update when camera position/orientation changes