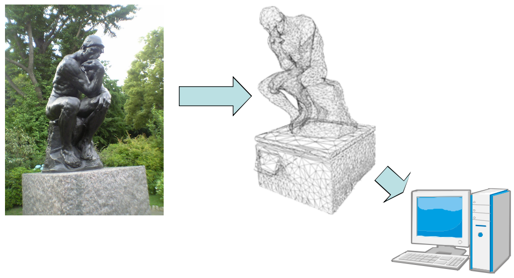
Data Visualization

Yasuhide Okamoto

Computer Vision

- Real scenes, objects -> Digital data



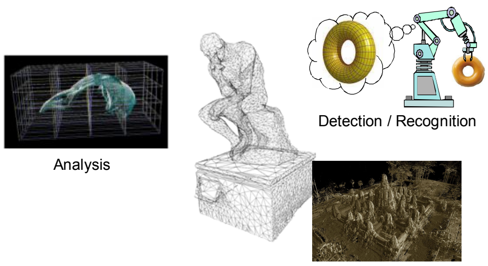
Utilization of 3D data

- What is obtained 3D data to be used for?

?

Utilization of 3D data

- What is obtained 3D data to be used for?

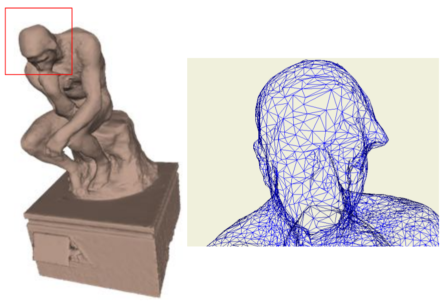


Utilization of 3D data

- How are they used?

?

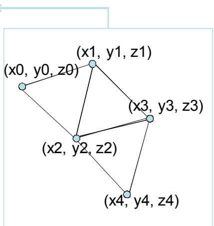
3D data format



3D data format

#3D data format
#Vertex List
(x_0, y_0, z_0)
(x_1, y_1, z_1)
(x_2, y_2, z_2)
…

#Face List
(0, 1, 2)
(1, 2, 3)
(1, 3, 4)
…

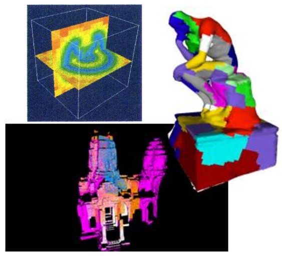


Difficult to understand by human…

Data visualization

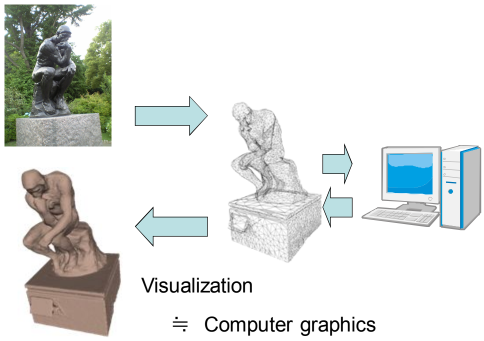
#3D data format
#Vertex List
(x_0, y_0, z_0)
(x_1, y_1, z_1)
(x_2, y_2, z_2)
…

#Face List
(0, 1, 2)
(1, 2, 3)
(1, 3, 4)
…



Easy to understand by human!

Next Step

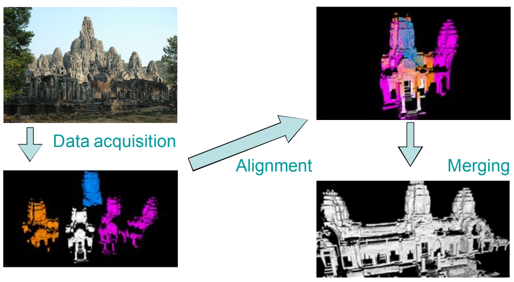


Visualization

≒ Computer graphics

Visualization for Huge 3D Data

Modeling from real objects



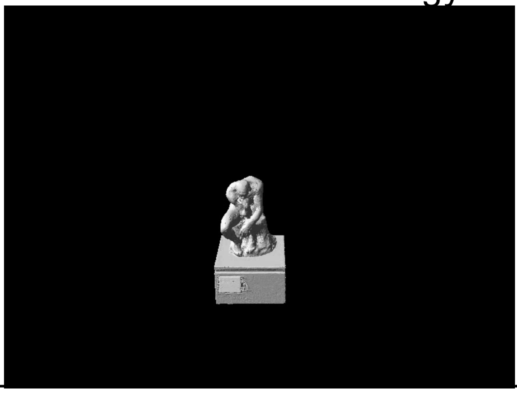
Latest 3D sensing technology

Leica ScanStation

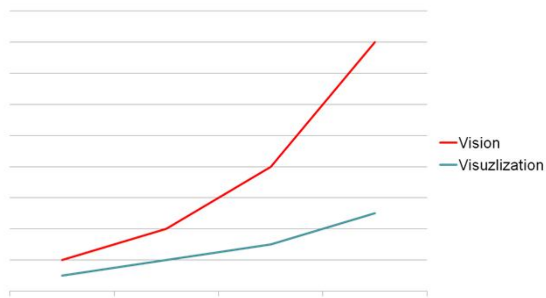
Microsoft Kinect

→ Easy to obtain fine and huge 3D data

Visualization technology



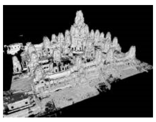
Improvement of vision / visualization ability



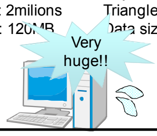
Problem

- Huge and fine 3D data cannot be used on commodity computers because of the size

Nara buddha
Triangles: 2milions
Data size: 120MB



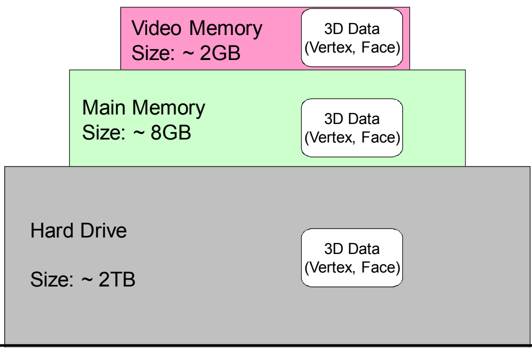
Bayon temple
Triangles: 1bilions
Data size: 100GB



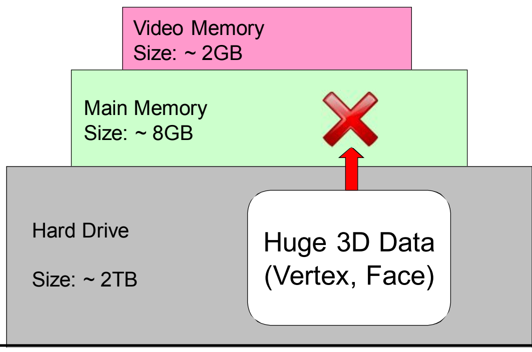
Content

- Simplification using Quadric error metrics
- Progressive Meshes
- Adaptive Tetrapuzzles
- Point based rendering (QSplat)

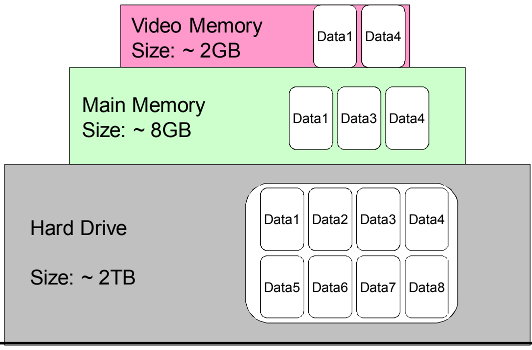
Memory hierarchy



Memory limitation

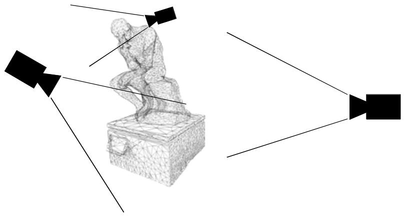


Out of Core



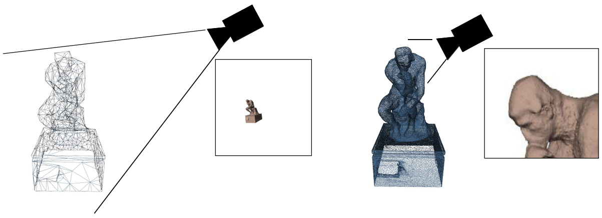
Difficulty in real time rendering

- Amount of data access pattern is too high



Level of Detail (LOD)

- Far side -> using rough model
- Near side -> using fine model



Level of Detail (LOD)

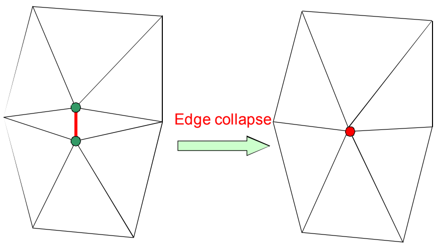
- Multilevel resolution model

Data size: large

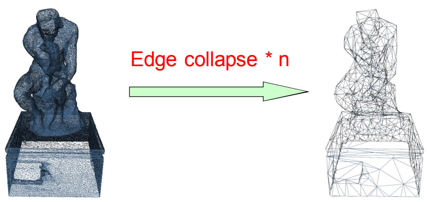
Data size: small

Mesh simplification

- Edge collapse



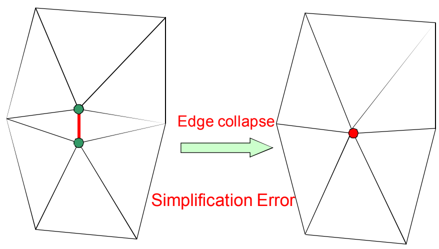
Mesh simplification



Surface simplification using Quadric Error Metrics [Garland97]

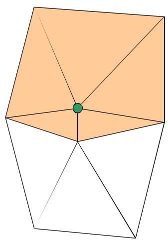
- Simplification method
  - with high quality approximations
  - can work efficiently
  - supporting highly complex objects

Define error of edge collapse



Error Metric

- Each vertex is the intersection of a set of planes



Error Metric

- Define the error at each vertex to be the sum of the squared distances to planes

$$\Delta(\mathbf{v}) = \Delta([v_x\ v_y\ v_z\ 1]^\mathsf{T}) = \sum_{\mathbf{p}\in \text{planes}(\mathbf{v})} (\mathbf{p}^\mathsf{T}\mathbf{v})^2$$

Where $p = [a\, b\, c\, d]^T$ represents the plane $ax + by + cz + d = 0$ with $a^2 + b^2 + c^2 = 1$

Error Metric

$$\begin{aligned}\Delta(v) &= \sum_{p \in planes(v)} (p^T v)^2 \\ &= \sum_{p \in planes(v)} (p^T v)^T (p^T v) \\ &= \sum_{p \in planes(v)} (v^T p)(p^T v) \\ &= \sum_{p \in planes(v)} v^T (pp^T) v \\ &= v^T \left(\sum_{p \in planes(v)} (pp^T) \right) v\end{aligned}$$

Error Metric

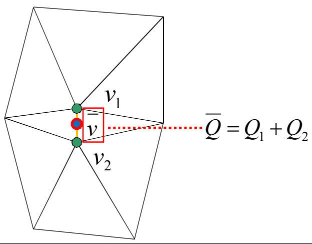
$$\begin{aligned}\Delta(v) &= v^T \left(\sum_{p \in planes(v)} (pp^T) \right) v \\ &= v^T \left(\sum_{p \in planes(v)} K_p \right) v\end{aligned}$$

$$\text{Where } K_p = pp^T = \begin{bmatrix} a^2 & ab & ac & ad \\ ba & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix}$$

The Kp can be used to find the squared distance of any point in space to the p.
We can sum these Kp and represent an entire set of planes by a single matrix **Q.**

Error Metric

- For each vertex v_i store a 4x4 matrix Q_i
- For a edge $(v_1, v_2) \rightarrow \bar{v}$, let $\bar{Q} = Q_1 + Q_2$



Where is $\bar{v}$?

- Simple scheme
  - v_1 or v_2
  - $(v_1 + v_2)/2$
- $\bar{v}$ which minimizes $\Delta(\bar{v})$

More on Quadrics

$$v_h = [v_x\, v_y\, v_z\, 1]^T, p = [a\, b\, c\, d]^T$$

$$\begin{aligned} D^2(v_h) = (p^T v_h)^2 &= (n^T v + d)^2 \quad where\ n = [a\, b\, c]^T \\ &= (v^T n + d)(n^T v + d) \\ &= (v^T n n^T v + 2dn^T v + d^2) \\ &= (v^T (nn^T) v + 2(dn)^T v + d^2) \end{aligned}$$

$$X = nn^T = \begin{bmatrix} a^2 & ab & ac \\ ba & b^2 & bc \\ ac & bc & c^2 \end{bmatrix} \quad y = dn = [da\, db\, dc]^T \quad z = d^2$$

More on Quadrics

$$Q = \begin{bmatrix} a^2 & ab & ac & ad \\ ba & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix} = Q(X, y, z)$$

$$X = nn^T = \begin{bmatrix} a^2 & ab & ac \\ ba & b^2 & bc \\ ac & bc & c^2 \end{bmatrix} \quad y = dn = [da\ db\ dc]^T \quad z = d^2$$

$$\Delta(v) = v^T Q v = v^T X v + 2y^T v + z$$

Optimum $\bar{v}$

$$\text{specify minimum } \Delta(\bar{v}) \rightarrow \nabla(\Delta(\bar{v})) = 0$$

$$\nabla(\Delta(\bar{v})) = 2X\bar{v} + 2y$$

$$2X\bar{v} + 2y = 0 \Rightarrow \bar{v} = -X^{-1} y$$

The associated minimum error is :

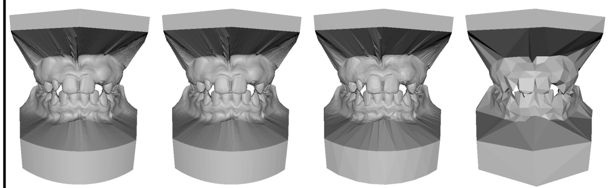
$$\Delta(\bar{v}) = y^T \bar{v} + z = -y^T X^{-1} y + z$$

Algorithm

1. Compute initial Q_i for all vertices
2. Compute v_i and $\bar{Q}$ for all edges
3. Search the edge with least error,
   Remove the edge,
   and Update errors around the edge
4. Iteratively do 3

Results

Results



Summary of QEM

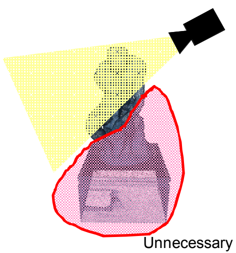
- Simplification using quadric error metric
  - High approximation
  - High efficiency
- QSlim
  - http://mgarland.org/software/qslim.html

Difficulty in fixed level LOD

- Transition of resolution is not smooth

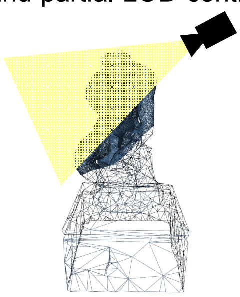
Difficulty in fixed level LOD

- Too fine triangles outside screen must be rendered when the camera is near-side.



Solution

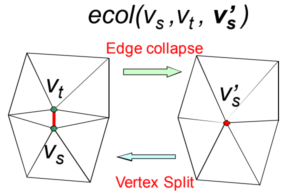
- Smooth, and partial LOD control



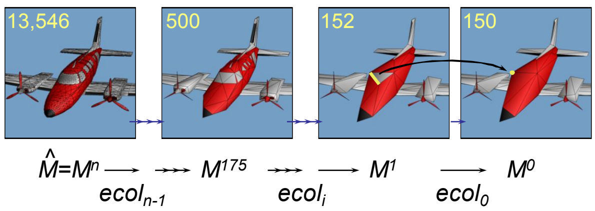
Progressive Meshes [Hoppe96]

- Simplification method
  - Lossless
  - Continuous resolution
  - Progressive
  - Using edge collapse and vertex split

Recording the sequence of edge collapses

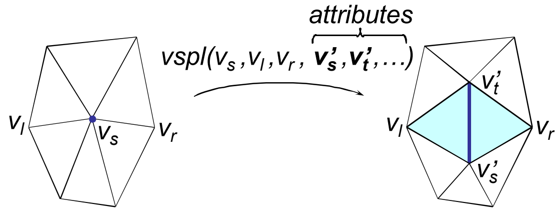


Recording the sequence of edge collapses

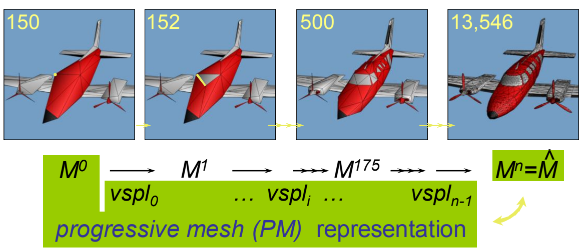


Edge collapse is invertible

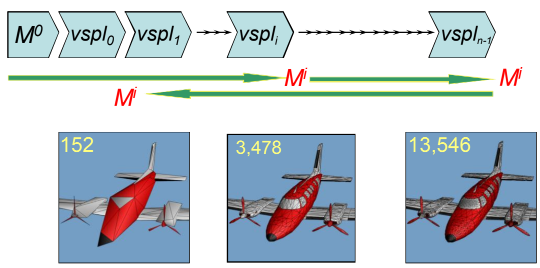
Vertex split transformation:



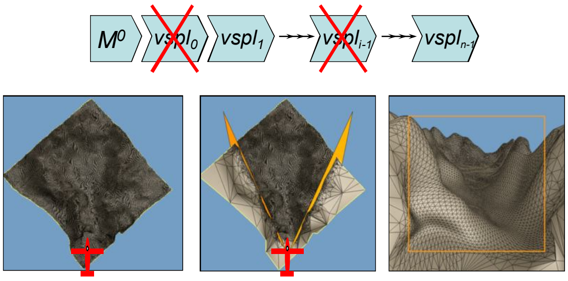
Reconstruction process



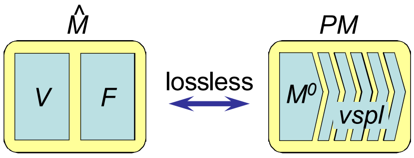
Continuous LOD



Selective refinement



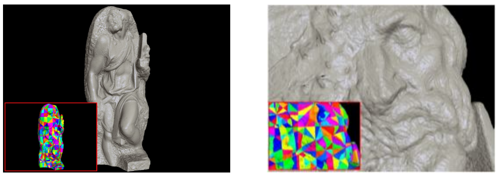
Summary of PM



- single resolution
- continuous-resolution
- smooth LOD
- space-efficient
- progressive

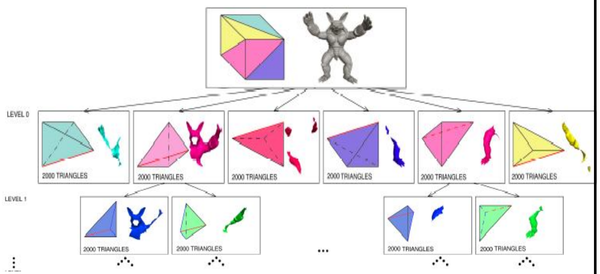
Adaptive Tetrapuzzles [Cignoni 04]

- Patch based LOD method
  - Not triangle based
    - Can reduce the cost of refine and simplification

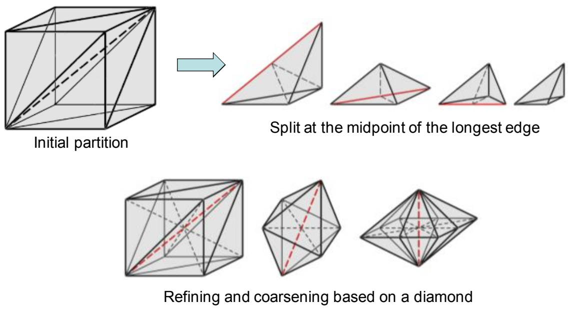


Multiresolution structure

- Recursive volumetric subdivision by tetrahedra

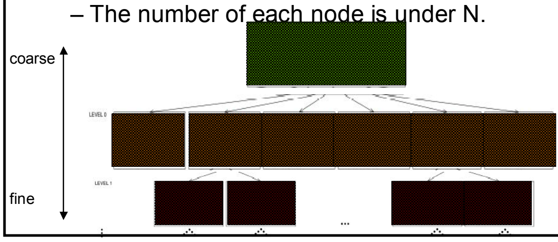


Tetrahedral partitioning



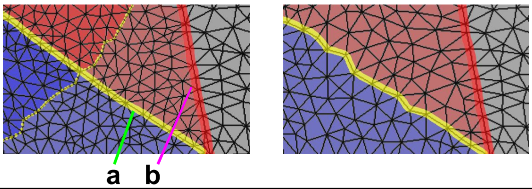
Simplification

- Simplification in bottom up order after recursive partitioning
- Simplification using quadric error metrics
  - The number of each node is under N.

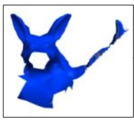


Constraint in simplification

- Constrains simplification on borders
  a. Diamond-internal borders
  b. Diamond-external borders
  c. Original borders



Data format



Patch parameters
- Bounding Sphere
  center(x, y, z), radius r
- Normal cone
  axis (nx, ny, nz), width θ
- Simplification error ε
  $\varepsilon = s\sqrt[3]{\varepsilon_q}$ where ε_q is QEM
- Pointer to children

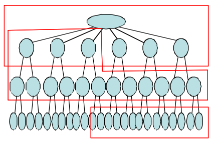
Geometric information
- Vertex array
- Index array

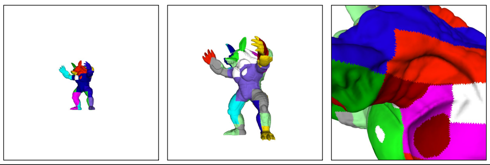
Data format

- Geometric data compression
  - Quantize vertices' parameters
    - Vertex position (px, py, pz) -> 24bit * 3
    - Normal vector (nx, ny, nz) -> 32bit
  - Index parameters
    - Triangle strips compression [Isenburg01]

View-dependent rendering

- Traverse, select, and render patches





Rendering algorithm

- Traverse hierarchy recursively

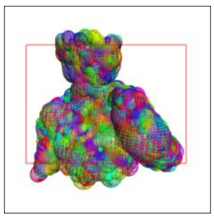
frustum / backface culling

Patch rendering

if (patch not visible)
    Skip this branch
else if (leaf node)
    Draw a patch
else if (εsize on screen (ε / v) < threshold)
    Draw a patch
else
    Traverse children

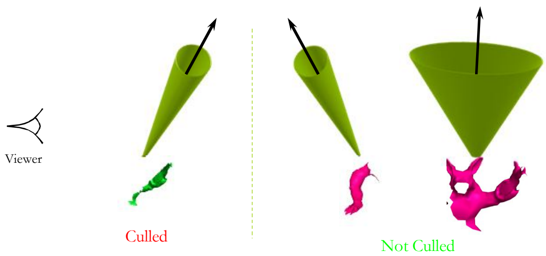
Frustum culling

- View-frustum culling
  - If the bounding sphere is out of screen, cull and backtrack.



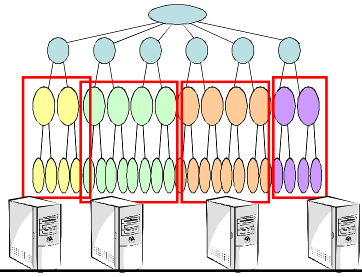
Backface culling

- If the cone faces entirely away from the viewer, cull and backtrack.



Details

- Parallel data construction
  - Partitioning and simplification of subtrees can be done independently

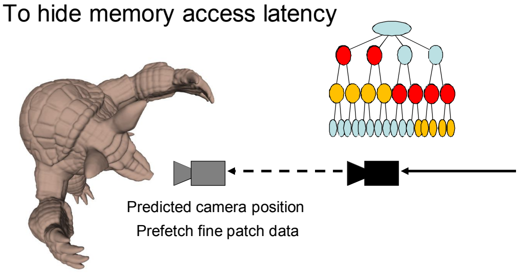


Details

- Memory coherency
  - Write patch data in the depth first order
  - rearrange triangles into the triangle strips
  - Memory management on VRAM by LRU strategy

Details

- Speculative prefetching
  - To hide memory access latency



Results

Model	Triangles	CPU	Data construction (sec)	Input size (MB)	Output size
Bonsai	6,317,116	2 15	1,741 219	289	76
David (2mm)	8,277,479	2 15	3,735 426	379	158
David (1mm)	56,230,343	2 15	24,499 3,594	2,574	967
St. Matthew	372,767,445	2 15	92,255 27,790	17,063	5,887

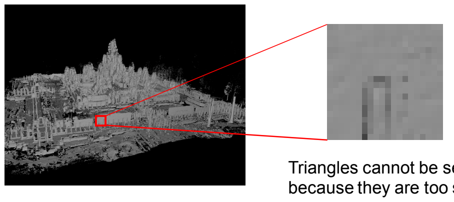
Demonstration video

Summary of ATP

- Patch-based LOD data structure
  - Small size of the hierarchy
  - Good approximation by constraints
  - Parallel data construction
- Some technical tunings
  - Memory coherency
  - Data compression
  - Prefetch

Another approaches

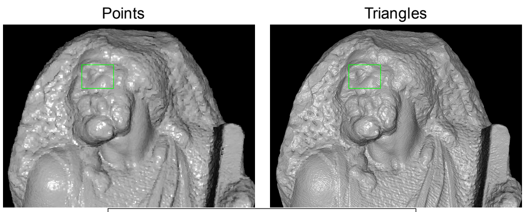
- Triangles are really necessary to render huge meshes?



Triangles cannot be seen because they are too small…

Point based rendering

- Use points instead of triangles



Not so different in aspects from far side

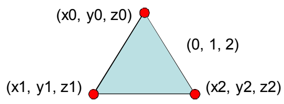
St. Matthew (127 millions triangles)

8 frame/sec 8 sec/frame

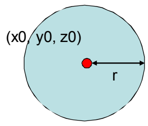
clearly different in rendering speed

Merits of point rendering

- Data simplicity
  - Triangles
    - 3 vertices
    - 1 set of indices

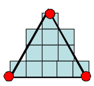


  - Points
    - 1 vertex
    - point size
    - (without connectivity)

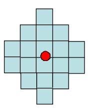


Merits of point rendering

- Processing cost
  - Triangles
    - Projection of 3 vertices
    - Precise rasterization

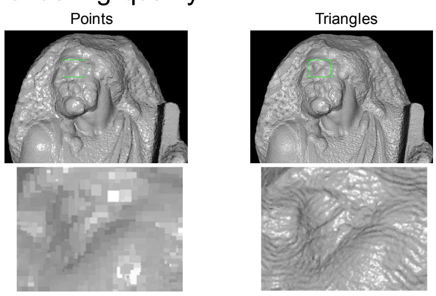


  - Points
    - Projection of 1 vertex
    - Simple rasterization



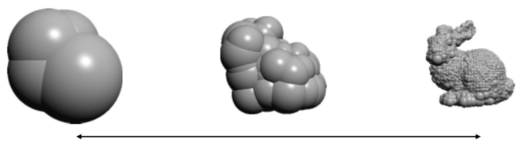
Demerit of point rendering

- Rendering quality



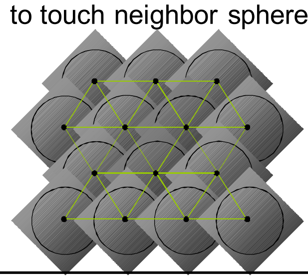
QSplat [Rusinkiewicz00]

- Point based rendering with LOD
  - Bounding sphere hierarchy



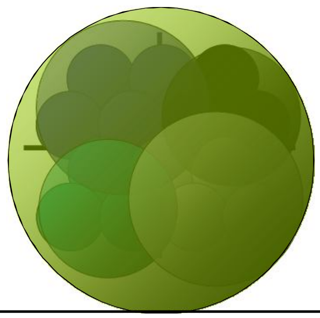
Data construction

- Input a triangle mesh
- Place a sphere at each triangle, large enough to touch neighbor spheres

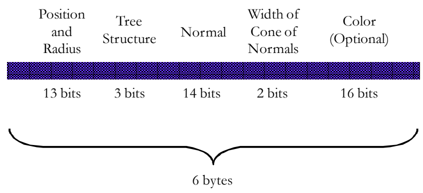


Creating a hierarchy

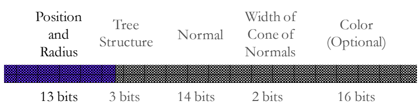
- Recursive splitting and merging



Node structure



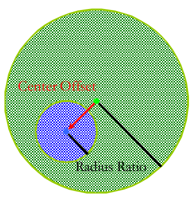
Position and radius



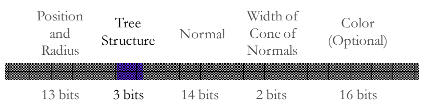
- Position and radius encoded relative to parent node

(x, y, z, r) are represented as $\frac{1}{13}$ to $\frac{13}{13}$

only 7621 combinations are valid, not 13^4

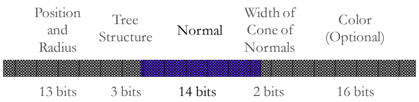


Tree structure

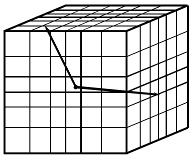


- Number of children (0, 2, 3, or 4) – 2 bits
- Presence of grandchildren – 1 bit

Normal

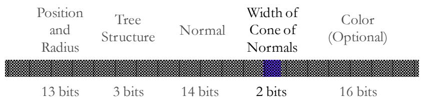


- Normal quantized to grid on faces of a cube



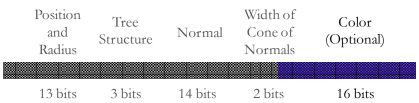
52×52×6

Normal cone



- Each node contains bounding cone of children's normals
- It is quantized to four values $\frac{1}{16}, \frac{4}{16}, \frac{9}{16}, \frac{16}{16}$

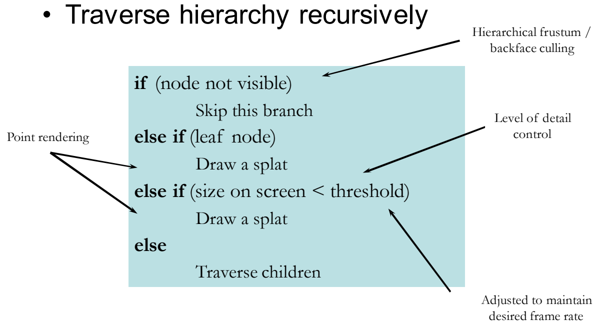
Color



- Per-vertex color is quantized 5-6-5 (R-G-B)

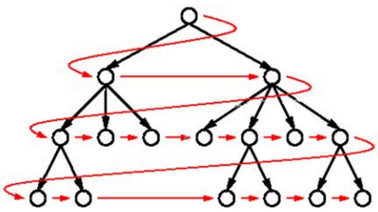
Rendering algorithm

- Traverse hierarchy recursively



Data alignment

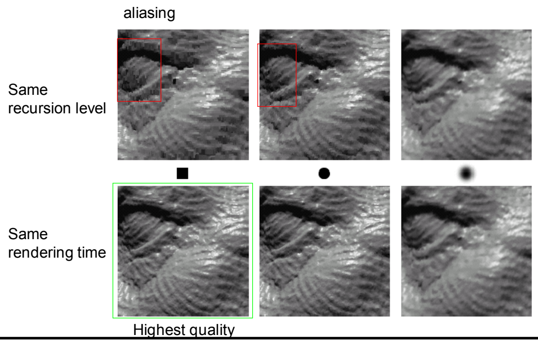
- Breadth-first order



Result

Model	Input points	Interior Nodes	Data construction (min)	Output size (MB)
David's head	2,000,651	974,114	0.7	18
David (2mm)	4,251,890	2,068,752	1.4	27
St. Matthew	127,072,827	50,285,122	59	761

Comparing splat shapes



Demonstration video

Summary of QSplat

- High speed and high quality rendering for complex models
- Fast preprocessing
- High compression rate