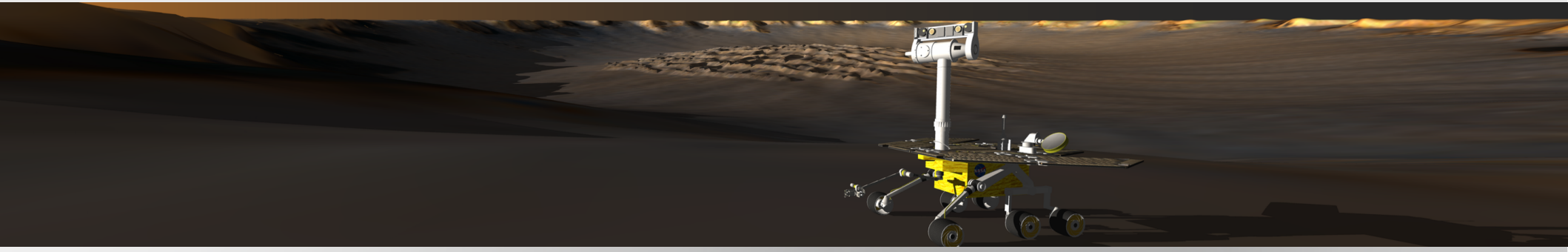
Accelerated Building and Rendering of Restricted BSP Trees

Brian Budge Daniel Coming Derek Norpchen Ken Joy

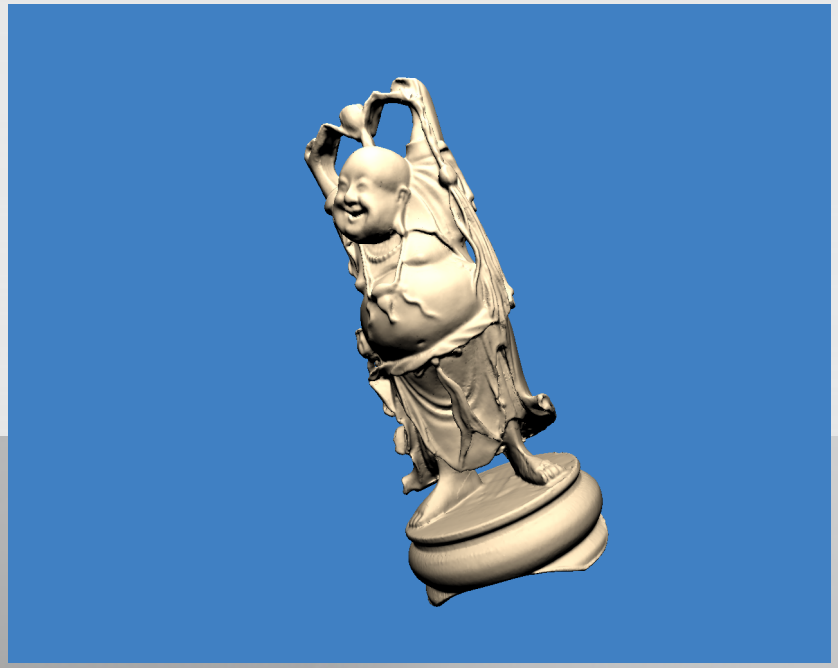
Institute for Data Analysis and Visualization
UC Davis

Center for Advanced Visualization,
Computation and Modeling
Desert Research Institute



Overview

- What is a Restricted BSP tree (RBSP)?
  - And what is the relation to *k*-DOPs?
- General Build Strategy for RBSP
- *k*-DOP details
  - *k*-DOP representation
  - Surface area calculations
- Ray tracing of RBSPs



Prior Work

- Lots of ray tracing acceleration work
- Havran's thesis [Havran 2000]
- Popular structures recently
  - *k*d-tree (example, [Wald 2006])
  - BVH (example, [Wald 2007])
  - BIH [Wächter 2006, Ooi 1990, Zachmann 2002]
- RBSP
  - Kammaje and Mora [Kammaje 2007]

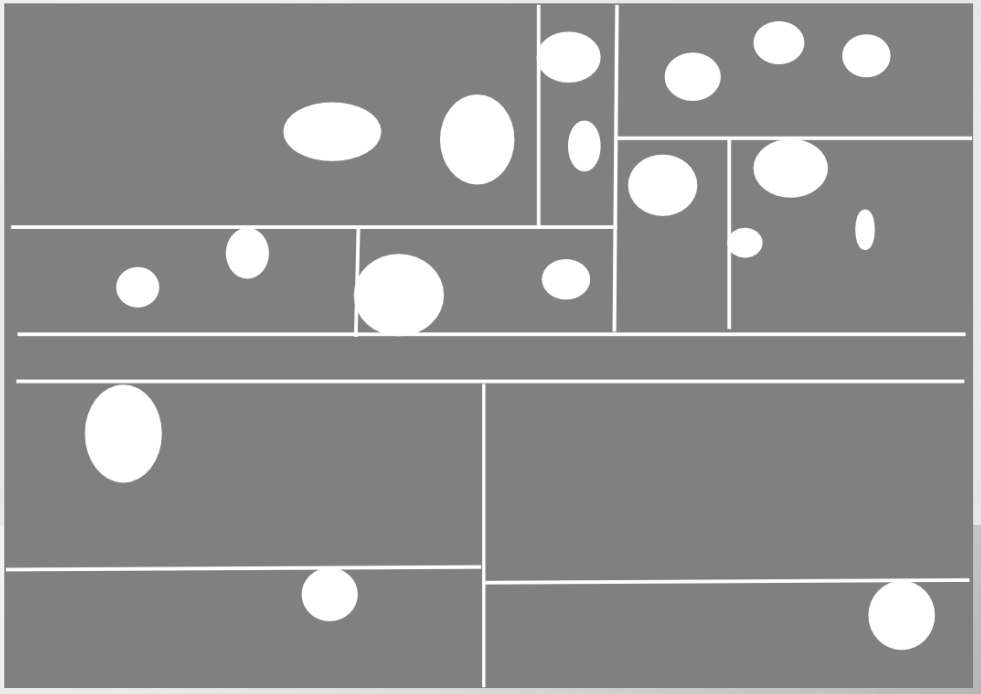
Why restricted BSP trees?

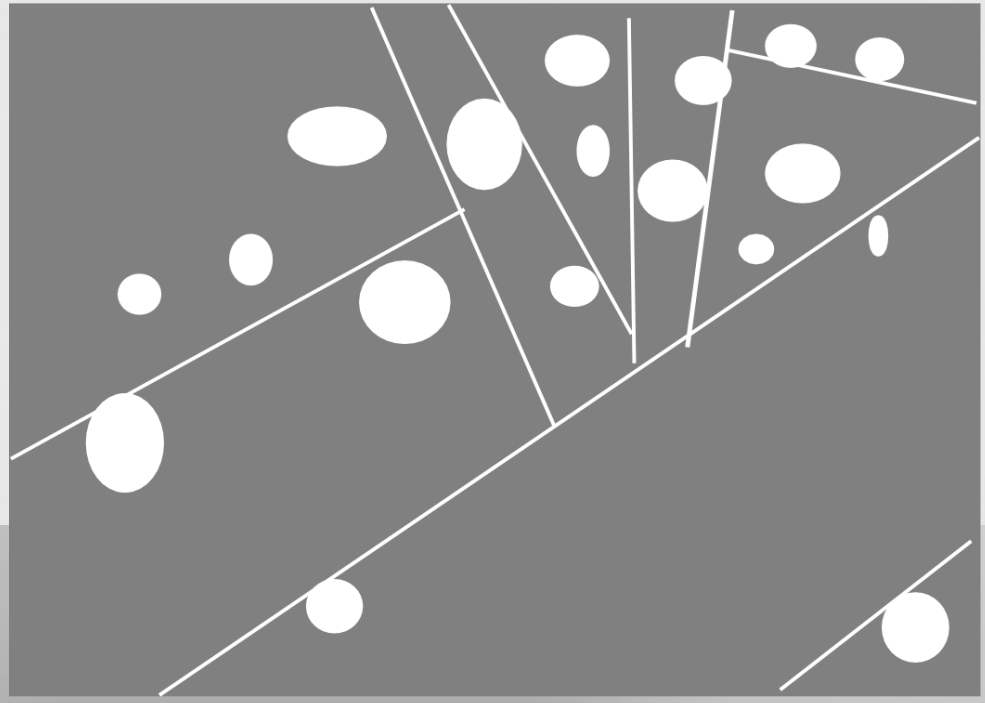
- *k*d-trees are so great, why not generalize?
- BSP trees can bound geometry better
- BSP trees require more work for traversal
- Not clear to us how to build good BSP tree
- RBSP: cross between *k*d-trees and BSP

What is an RBSP?

*k*d-tree splits axis-aligned bounding boxes(AABB)

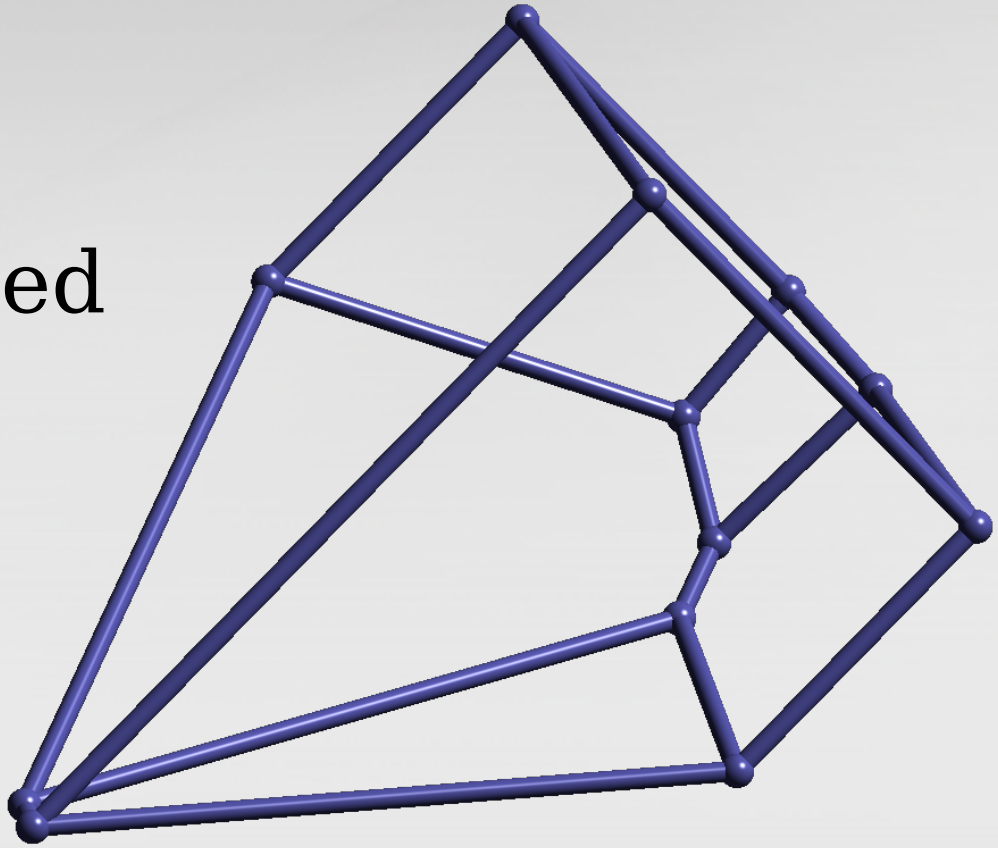
RBSP splits *k*-DOPs





What is a *k*-DOP?

- Convex polytope
- Defined by constrained set of M directions
  - $k = 2M$
- AABB = 6-DOP with directions x, y, z
- Can describe k-DOP with set of M directions and set of M intervals



Edge Soup

- Simplest explicit k-DOP form

```
struct edge {
    line_segment seg;
    int faceID[2]; // Just ID's, faces not stored anywhere
};
```

- Compact and efficient

k-DOPs we used

- Standard k-DOPs [Klosowski 1998]
  - Features of cube to generate 3, 7, and 13
  - We extended by cube feature subdiv for 25 and 49 directions (k=50, 98)
- Golden Ratio directions
  - Point locations generated on sphere in a spiral
  - Generates decent point distribution
  - What Kammaje and Mora used

RBSP build overview

- Build strategy similar to *k*d-tree
  - Binary space partitioning via split planes
  - Greedy
  - Recursive
- M split plane directions instead of 3

RBSP build overview

```
buildTreeNode(kDOP, objects) {
      if(terminate(kDOP, objects)) {
            setObjects(objects);
            return this;
      }
      info = findSplit(kDOP, objects);
      if(!info.goodSplit)  {
         setObjects(objects);
         return this;
      }
      split(info, kDOP, kDOP_r, objects, obj_r);
      left = buildTreeNode(kDOP, objects);
      right = buildTreeNode(kDOP_r, obj_r);
      return this;
}
```

RBSP build overview

```
buildTreeNode(kDOP, objects) {
      if(terminate(kDOP, objects)) {
            setObjects(objects);
            return this;
      }
      info = findSplit(kDOP, objects);
      if(!info.goodSplit)  {
         setObjects(objects);
         return this;
      }
      split(info, kDOP, kDOP_r, objects, obj_r);
      left = buildTreeNode(kDOP, objects);
      right = buildTreeNode(kDOP_r, obj_r);
      return this;
}
```

Termination of recursion

- Typical criteria
  - Minimal triangle count in the node
  - Maximum tree depth reached

RBSP build overview

```
buildTreeNode(kDOP, objects) {
      if(terminate(kDOP, objects)) {
            setObjects(objects);
            return this;
      }
      info = findSplit(kDOP, objects);
      if(!info.goodSplit)  {
         setObjects(objects);
         return this;
      }
      split(info, kDOP, kDOP_r, objects, obj_r);
      left = buildTreeNode(kDOP, objects);
      right = buildTreeNode(kDOP_r, obj_r);
      return this;
}
```

Finding a split plane

- Many different possible heuristics
- Only requirements:
  - The plane is an offset along one of the k-DOP directions
  - The plane intersects the current k-DOP
- Following prior work, we use SAH

Finding a split

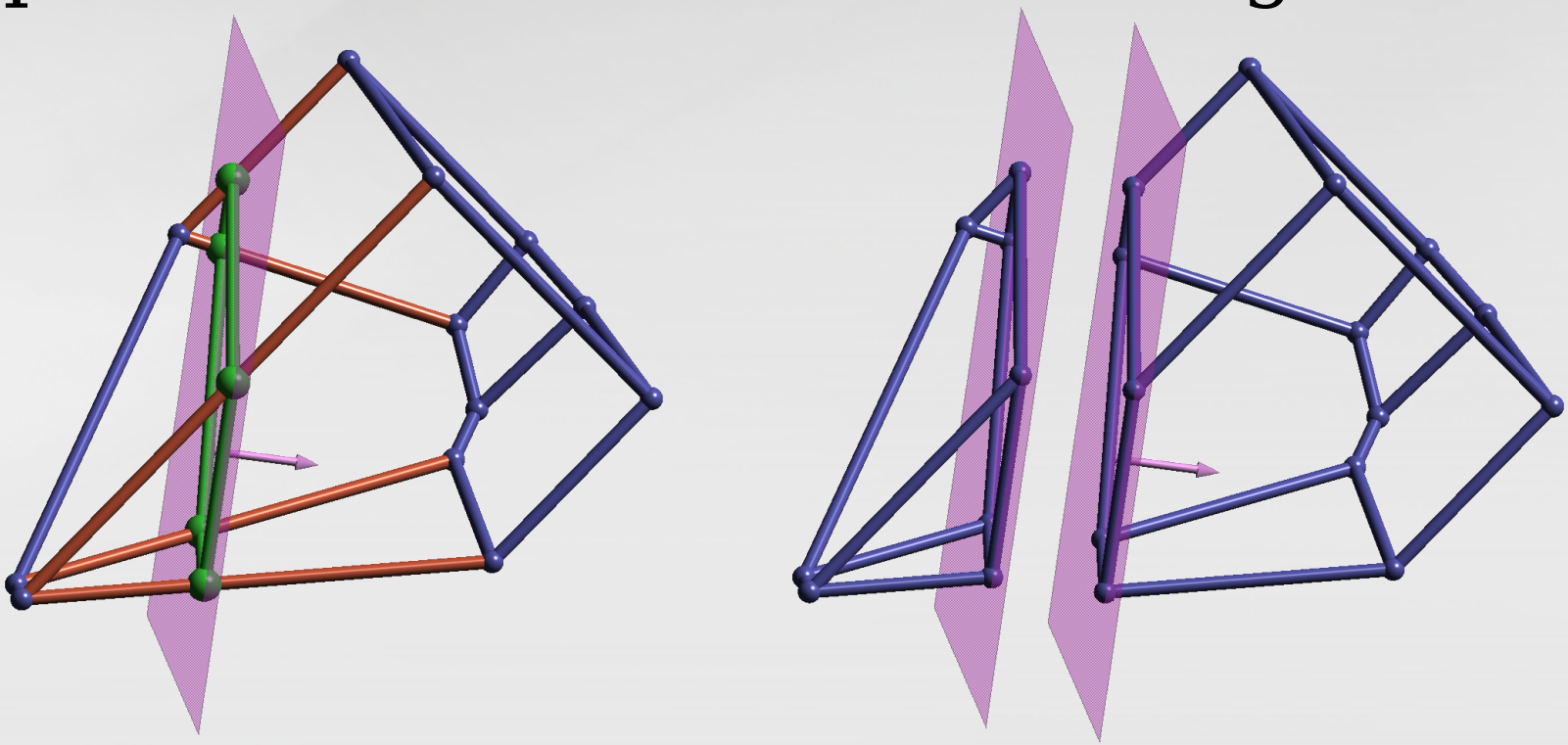
```
info.cost = c_intersect * len;
info.goodSplit = false;

estimator.init(kDOP);
for(m = 0; m < M; ++m) {
    estimator.setDirection(m);
    //// extract object bounds for direction m
    //// sort the bounds
    //go through and figure out the cost at each potential split
    for(size_t i = 0; i < 2*len; ++i) {
        (saRatioBelow, saRatioAbove) = estimator.areas(edges[i].t);
        costBelow = nBelow * saRatioBelow;
        costAbove = nAbove * saRatioAbove;
        cost = c_traverse + c_intersect*(costBelow + costAbove);
        if(cost < info.cost) {
            info.cost = cost;
            info.dir = m;
            info.split = edget;
            info.goodSplit = true;
        }
    }
}
return info;
```

RBSP build overview

```
buildTreeNode(kDOP, objects) {
      if(terminate(kDOP, objects)) {
            setObjects(objects);
            return this;
      }
      info = findSplit(kDOP, objects);
      if(!info.goodSplit)  {
         setObjects(objects);
         return this;
      }
      split(info, kDOP, kDOP_r, objects, obj_r);
      left = buildTreeNode(kDOP, objects);
      right = buildTreeNode(kDOP_r, obj_r);
      return this;
}
```

Splitting the node

- Split the k-DOP into left and right



- Split the objects left, right, or both
  - If both, can do perfect splitting

RBSP build overview

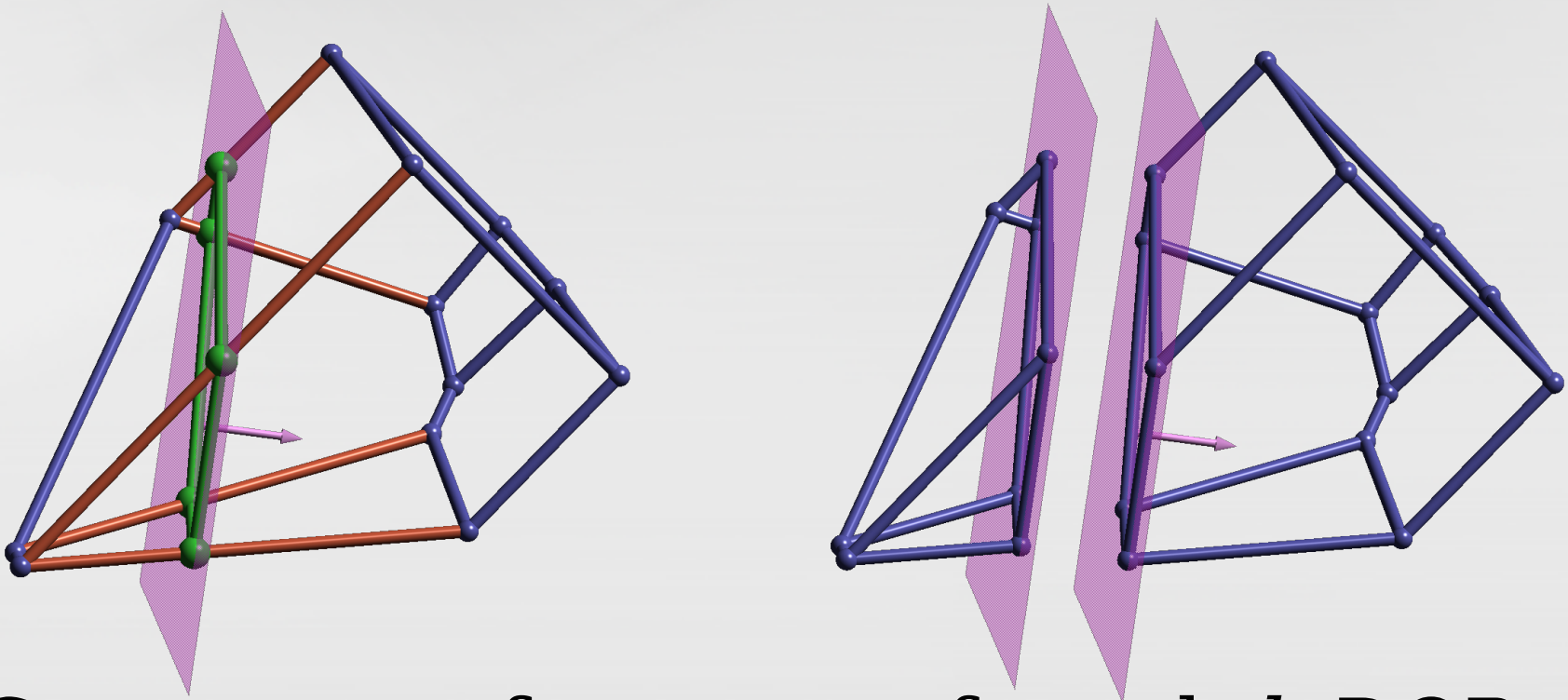
```
buildTreeNode(kDOP, objects) {
      if(terminate(kDOP, objects)) {
            setObjects(objects);
            return this;
      }
      info = findSplit(kDOP, objects);
      if(!info.goodSplit)  {
         setObjects(objects);
         return this;
      }
      split(info, kDOP, kDOP_r, objects, obj_r);
      left = buildTreeNode(kDOP, objects);
      right = buildTreeNode(kDOP_r, obj_r);
      return this;
}
```

Splitting Strategy: SAH

- Need surface area ratios for each possible split
- Surface area of k-DOP is not cheap
- Many splits considered
- Strategy: small precomputation yields cheap evaluation per split candidate

Naïve SAH on *k*-DOPs

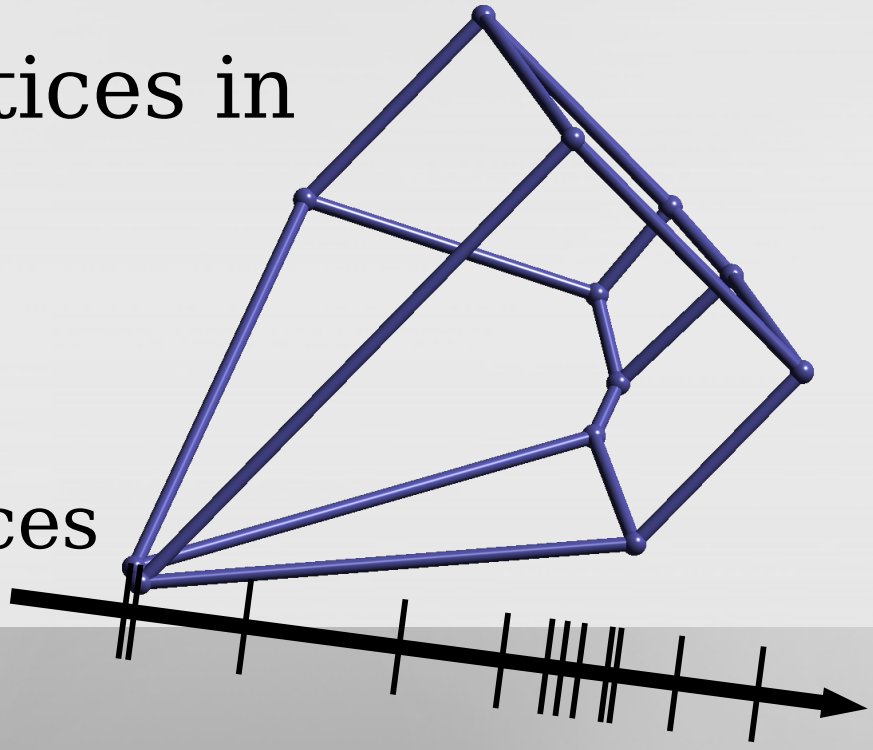
- Explicitly split *k*-DOP with candidate plane



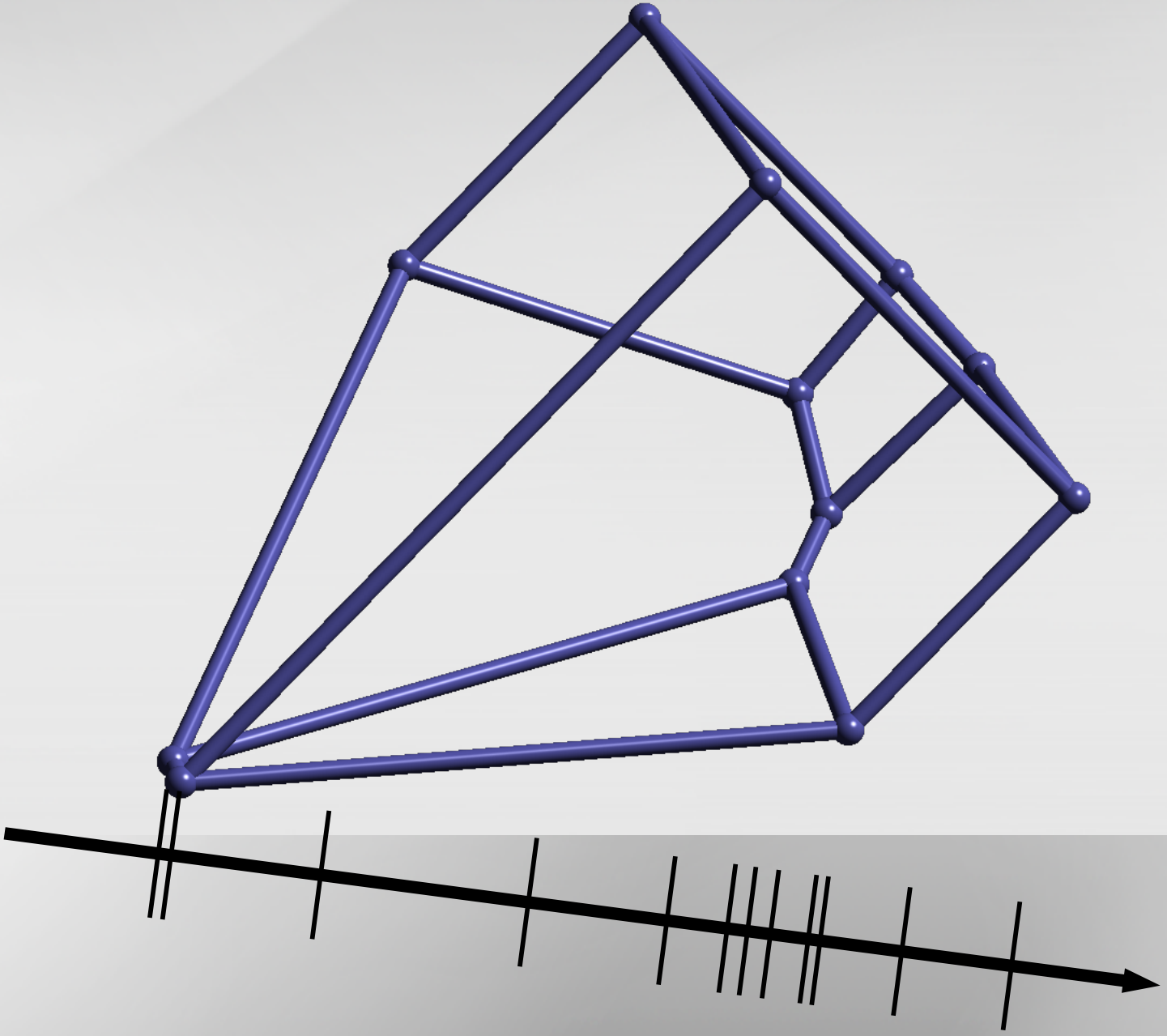
- Compute surface area of result *k*-DOPs
  - Polygon area

Dynamic Programming SAH on *k*-DOPs

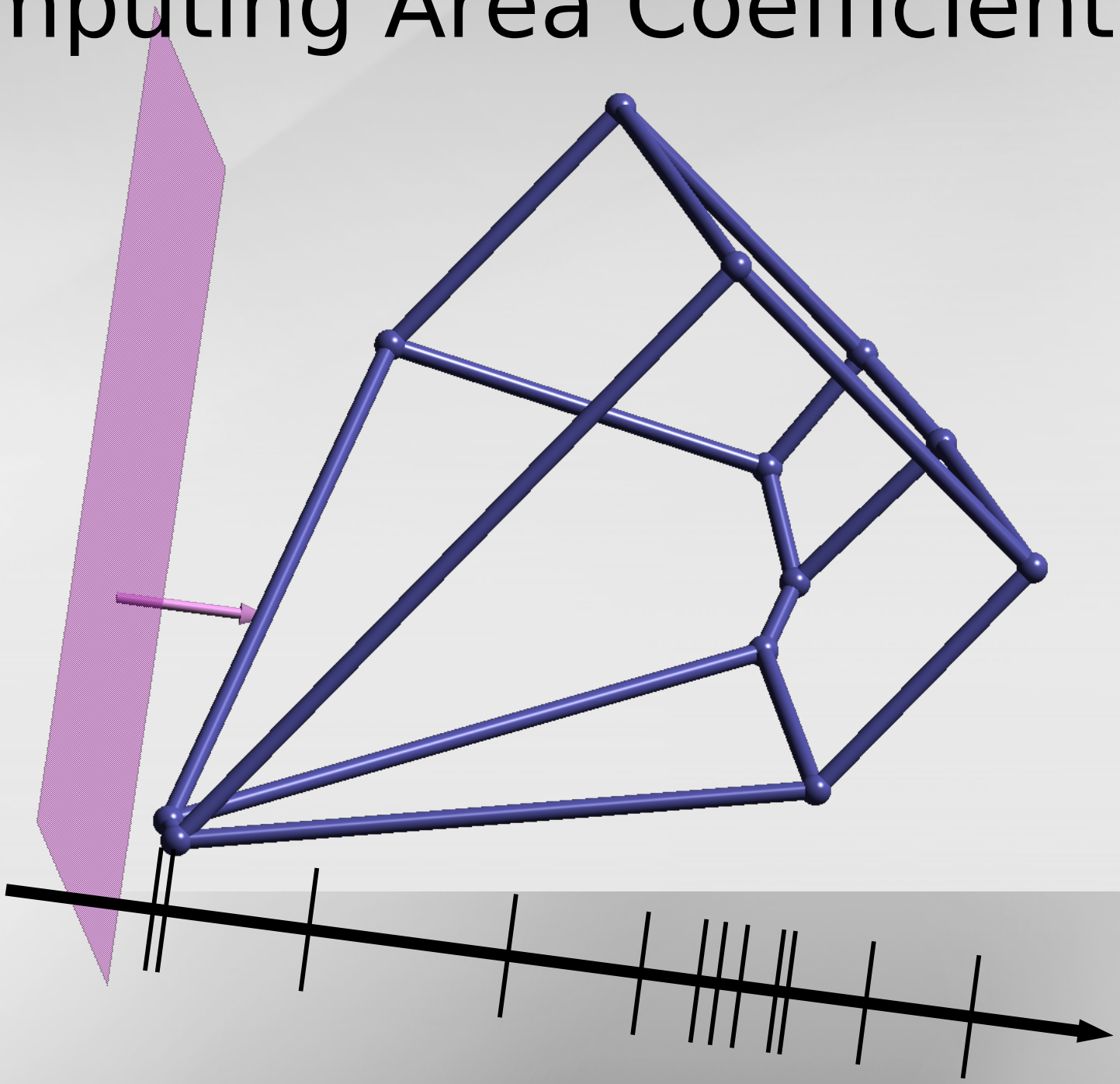
- Want parameterized area equations
  - 2nd order polynomials
  - $Area_{left}(t)$, $Area_{split}(t)$
- Project/sort *k*-DOP vertices in splitting direction
  - Each face contributes to coefficients
  - Intervals between vertices use same coefficients



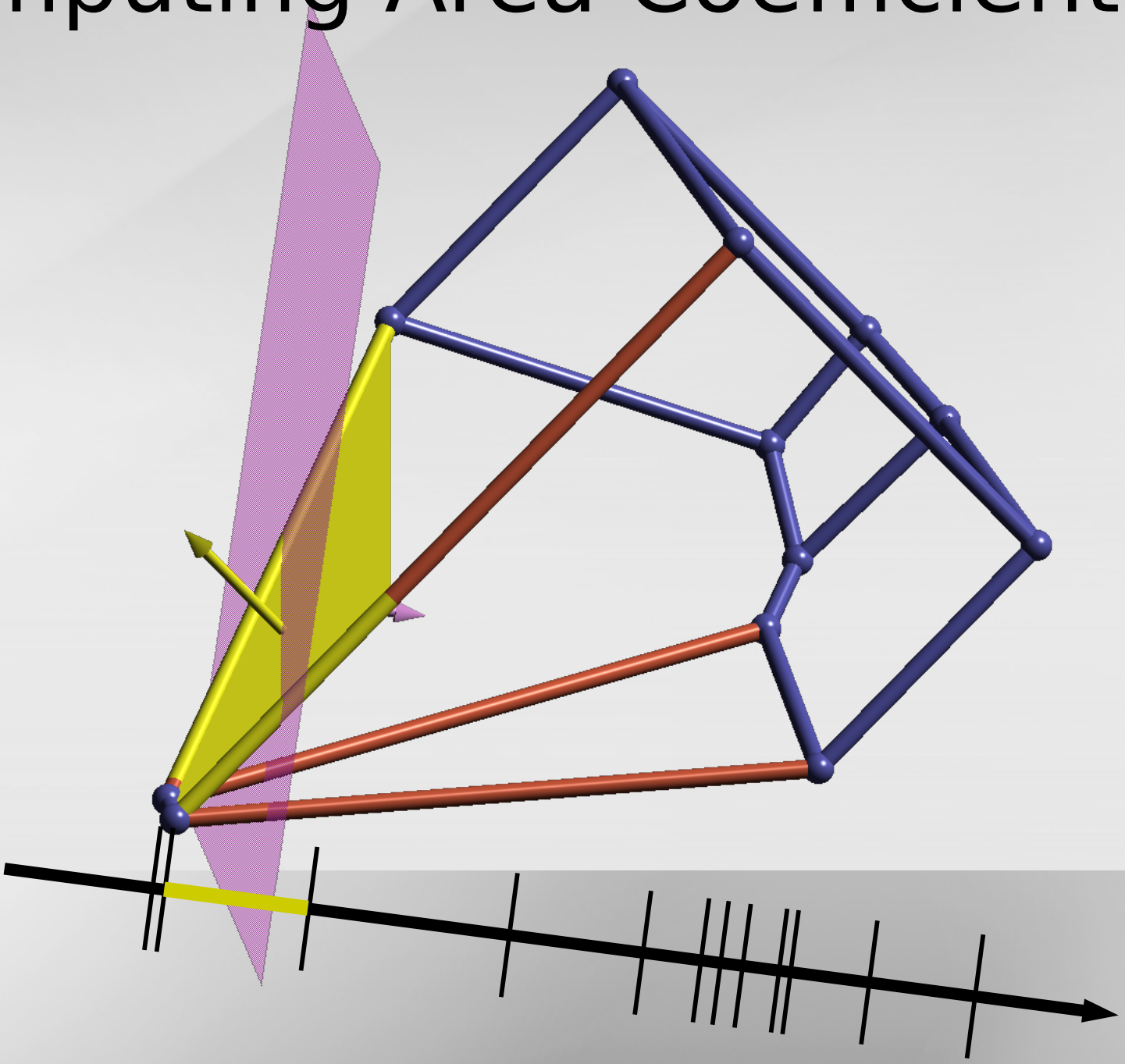
Computing Area Coefficients



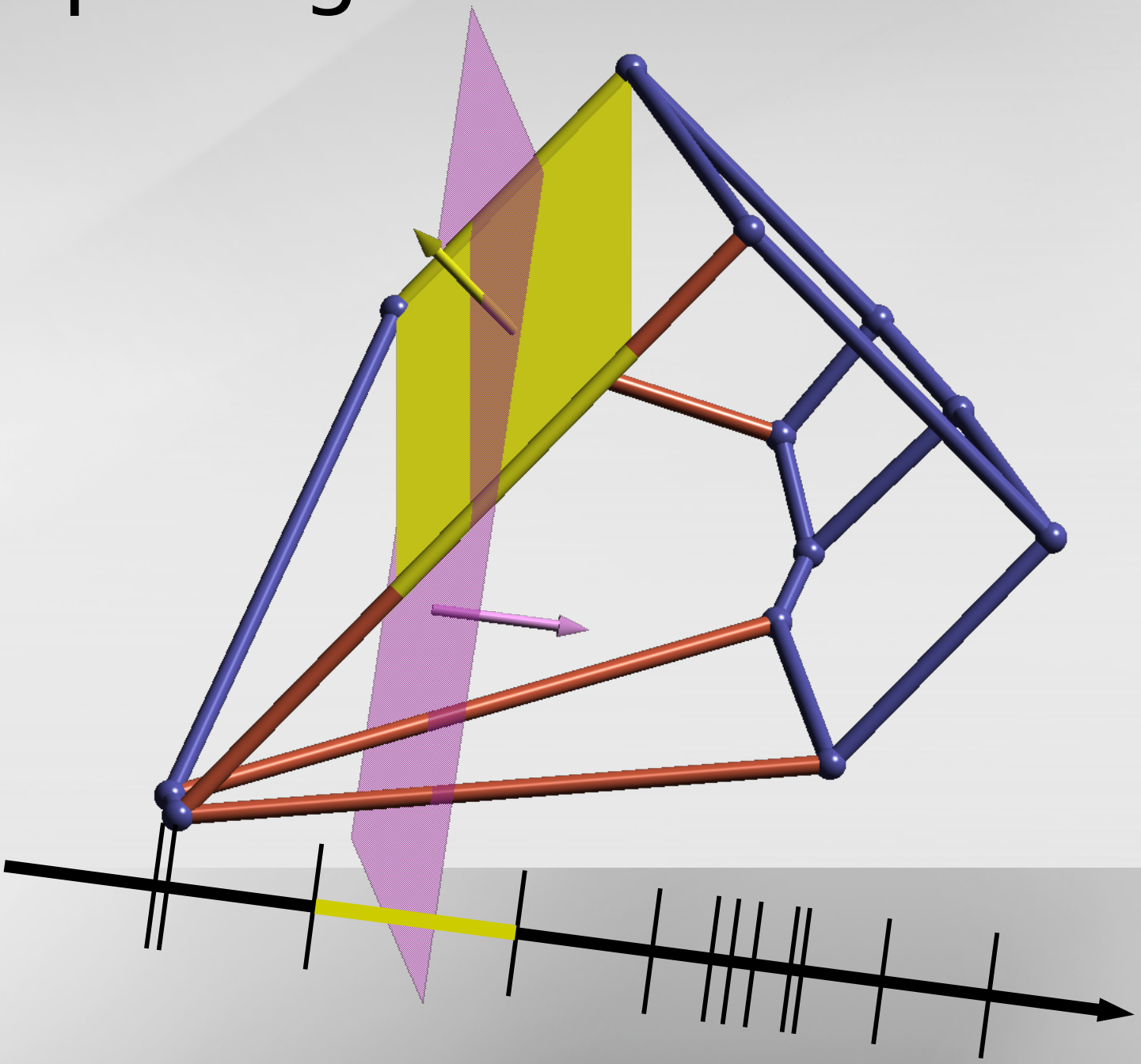
Computing Area Coefficients



Computing Area Coefficients



Computing Area Coefficients

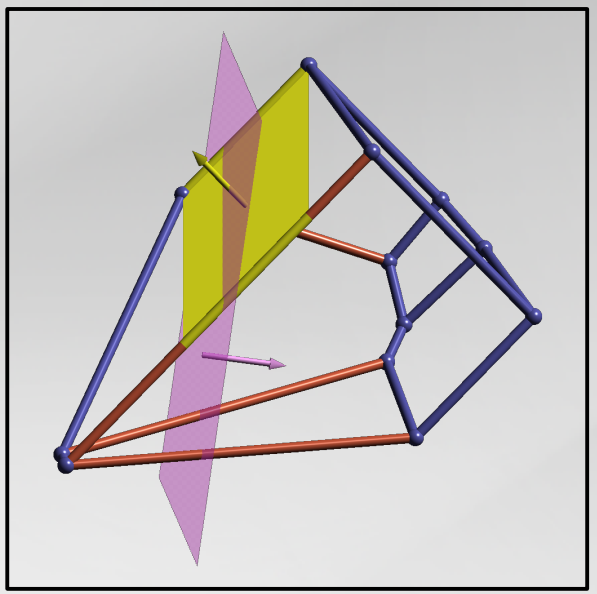


Coefficients for $\text{Area}_{left}(t)$

$$\frac{\delta Area_i}{\delta t} = 2c_2 t + c_1,$$

$$c_۲ = \frac{۱}{۲} \cdot \underbrace{\frac{h_j - h_i}{t_j - t_i}}_{\text{Rate of convergence/ divergence}} \cdot \underbrace{\frac{۱}{\sin(\cos^{-۱}(F \cdot D))}}_{\text{Account for angle between face and splitting direction}},$$

$$c_1 = \underbrace{\left(h_i - t_i \frac{h_j - h_i}{t_j - t_i}\right)}_{\text{Signed distance of edges at t = 0}} \cdot \frac{1}{\sin(\cos^{-1}(F \cdot D))}$$

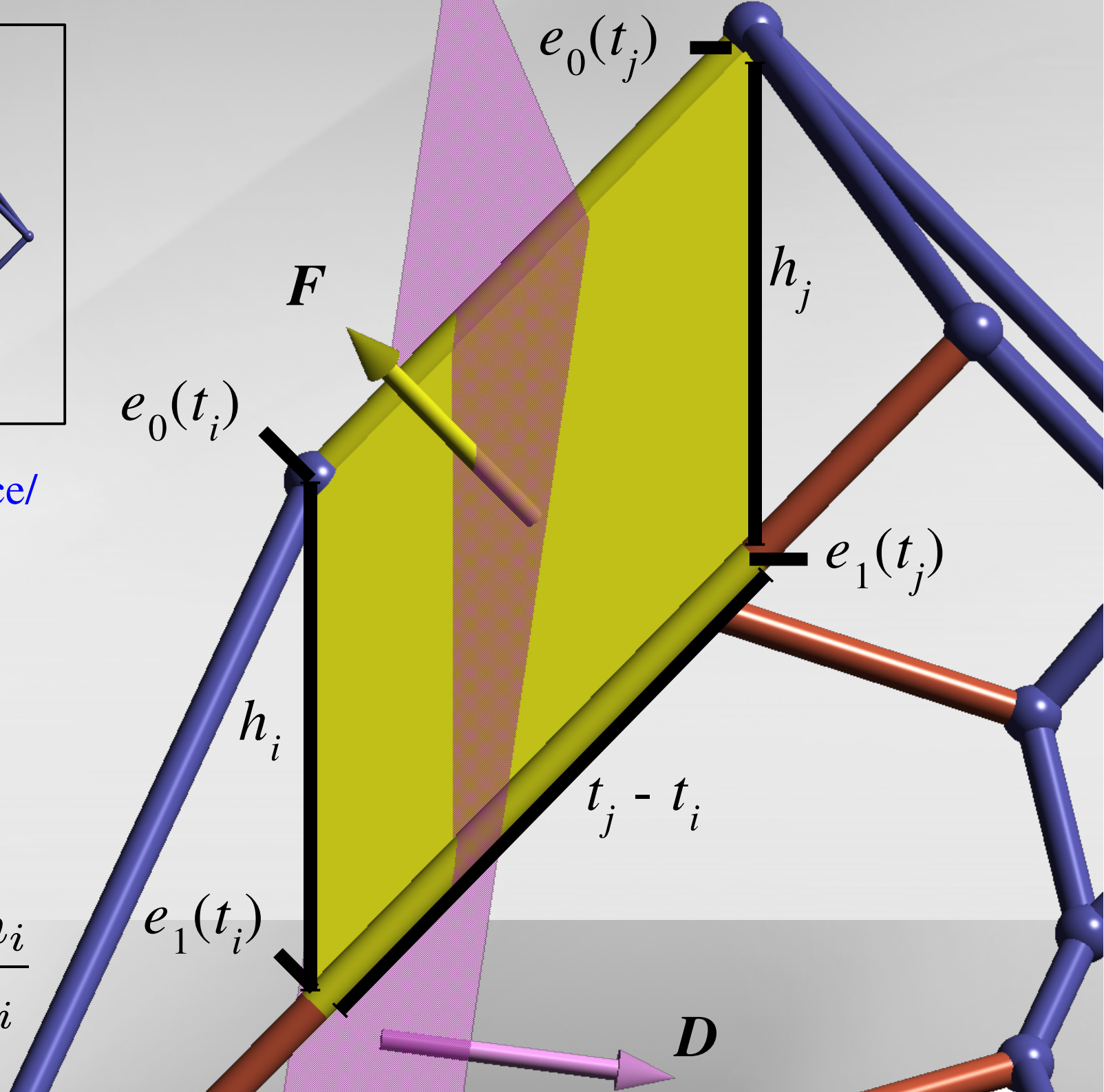


Rate of convergence/ divergence

$$\frac{h_j - h_i}{t_j - t_i}$$

Signed distance of edges at $t = 0$

$$h_i - t_i \frac{h_j - h_i}{t_j - t_i}$$



Dynamic Programming
$Area_{left}(t)$

- Single pass over sorted vertices
- Accumulate coefficients

$$C_2 = \sum_{i=0}^{E} c_{2i}, \qquad C_1 = \sum_{i=0}^{E} c_{1i},$$

- Accumulate partial sums of area completely to left

$$S_{i+1} = C_2(t_{i+1}^2 - t_i^2) + C_1(t_{i+1} - t_i) + S_i$$

$$t_i <= t < t_{i+1}$$

- Need to compute initial face area S_0

Dynamic Programming

$Area_{left}(t)$

- Dimension reduction [Sunday, 2002]:

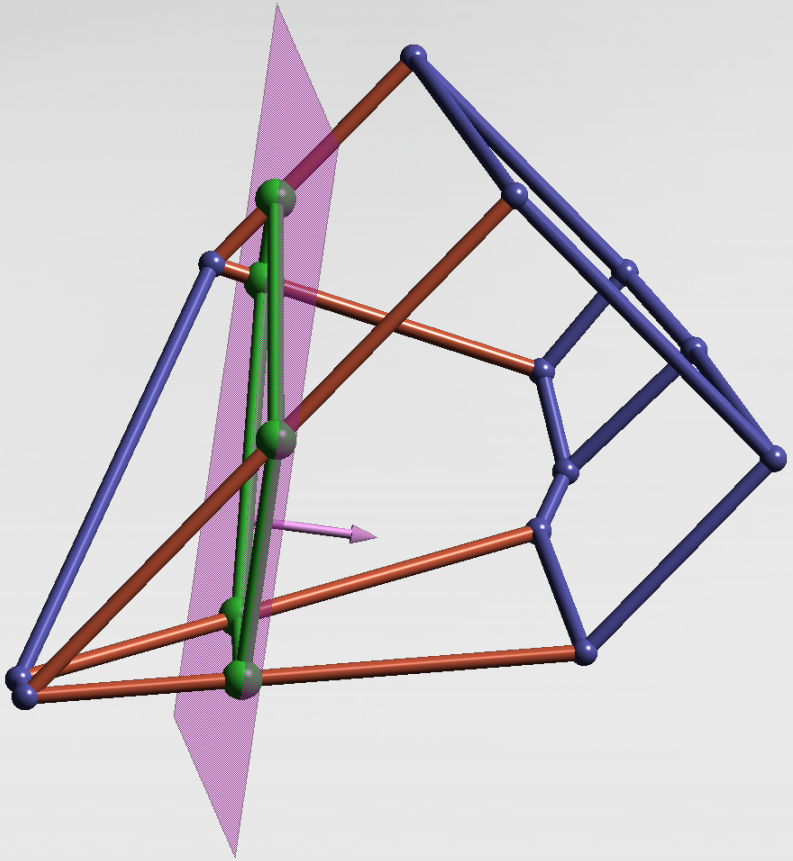
$$S_0 = Area3D = Area2D \cdot \frac{|D|}{\max(D_x, D_y, D_z)}$$

$$Area2D = \underbrace{\frac{1}{2}\sum_{i=0}^{E}(u_{i+1} + u_i)(v_{i+1} - v_i)}_{\text{Signed area of the triangles formed by each edge and the origin}},$$

$$u, v \in \{x, y, z\} \mid D_u, D_v \neq \max(D_x, D_y, D_z)$$

Coefficients for $Area_{split}(t)$

- Parameterize 2D polygon area in splitting direction
- Parameterize vertices of plane-k-DOP intersection along intersected edges
- Only need to know two adjacent vertices at once



$$Area2D_{\text{split}}(t) = \frac{1}{2}\sum_{i=0}^{E}(u_{i+1}(t) + u_i(t))(v_{i+1}(t) - v_i(t))$$

Dynamic Programming

$Area_{split}(t)$

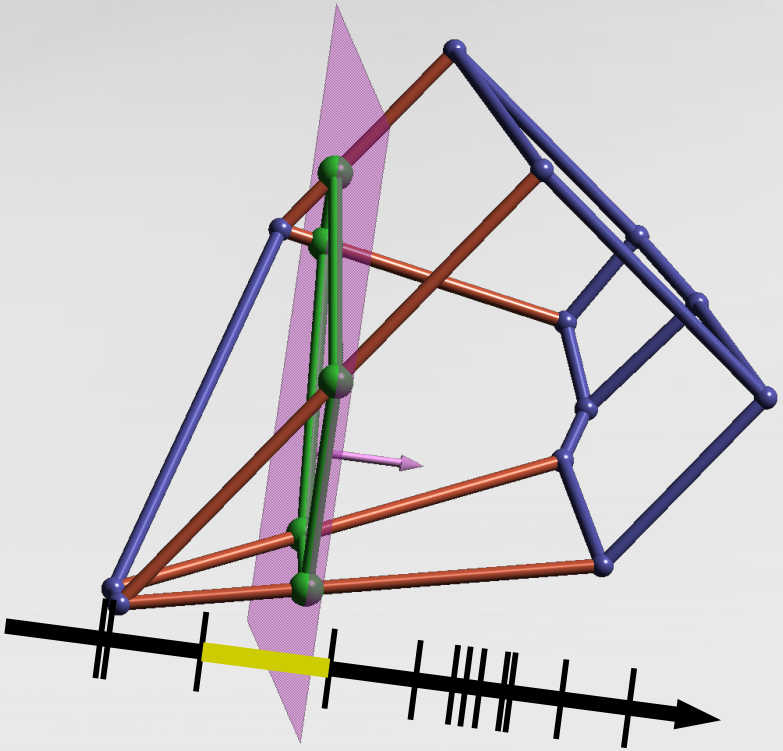
$$Area2D_{\text{split}}(t) = \frac{1}{2}\left(K_2 t^2 + K_1 t + K_0\right),$$

$$K_2 = \sum_{i=0}^{E} k_{2i}, \quad K_1 = \sum_{i=0}^{E} k_{1i}, \quad K_0 = \sum_{i=0}^{E} k_{0i}.$$

$$\begin{aligned}
k_{۲i} &= (d_{u,i+۱} + d_{u,i})(d_{v,i+۱} - d_{vi}),\\
k_{1i} &= (d_{u,i+1} + d_{u,i})(o_{v,i+1} - o_{vi})\\
&\quad +(o_{u,i+1} + o_{u,i})(d_{v,i+1} - d_{vi}),\\
k_{0i} &= (o_{u,i+1} + o_{u,i})(o_{v,i+1} - o_{vi}).
\end{aligned}$$

Computing Areas for Splits

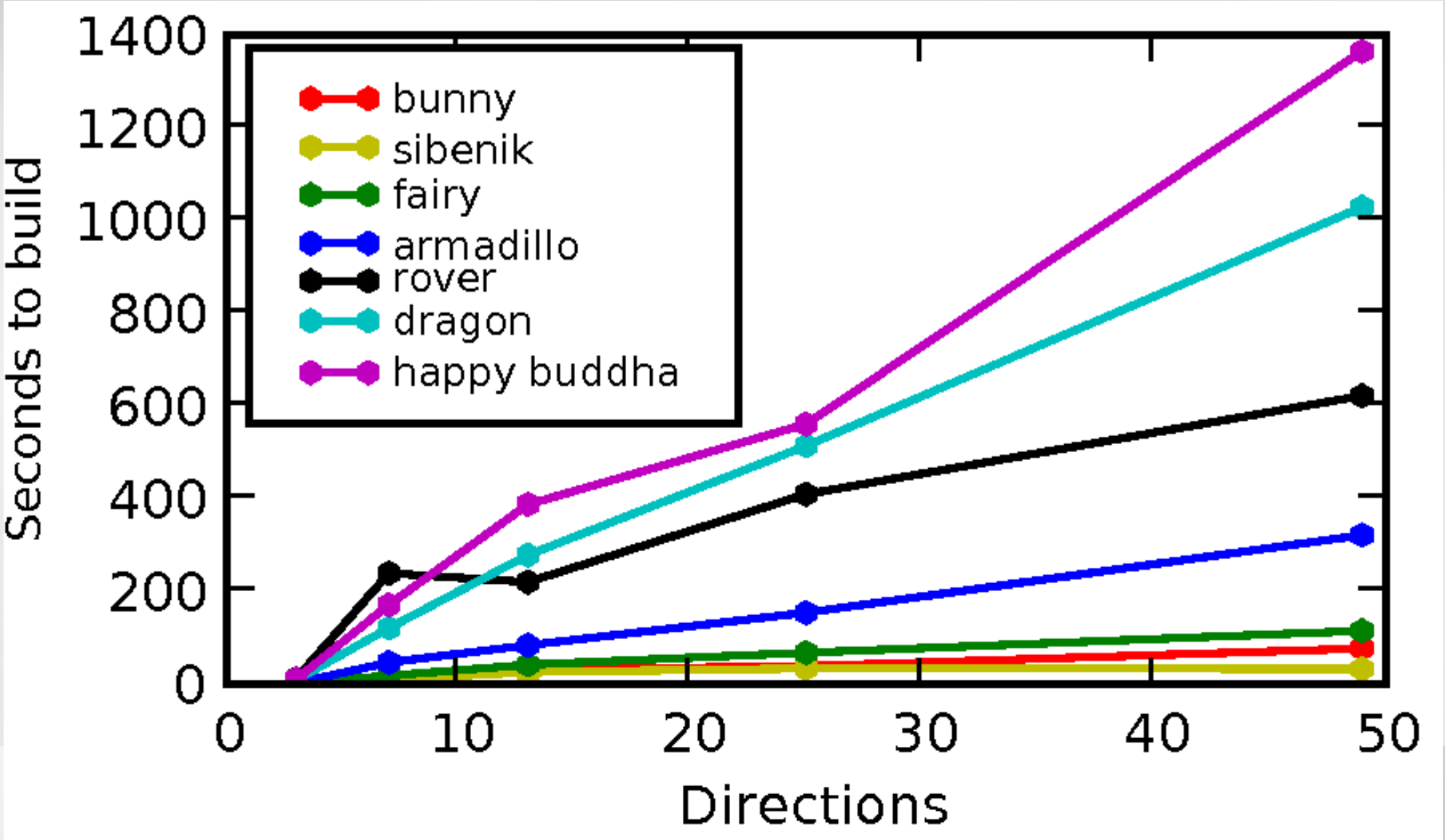
- Given split candidate offset
- Find coefficients for interval intersected by candidate
- Compute areas:



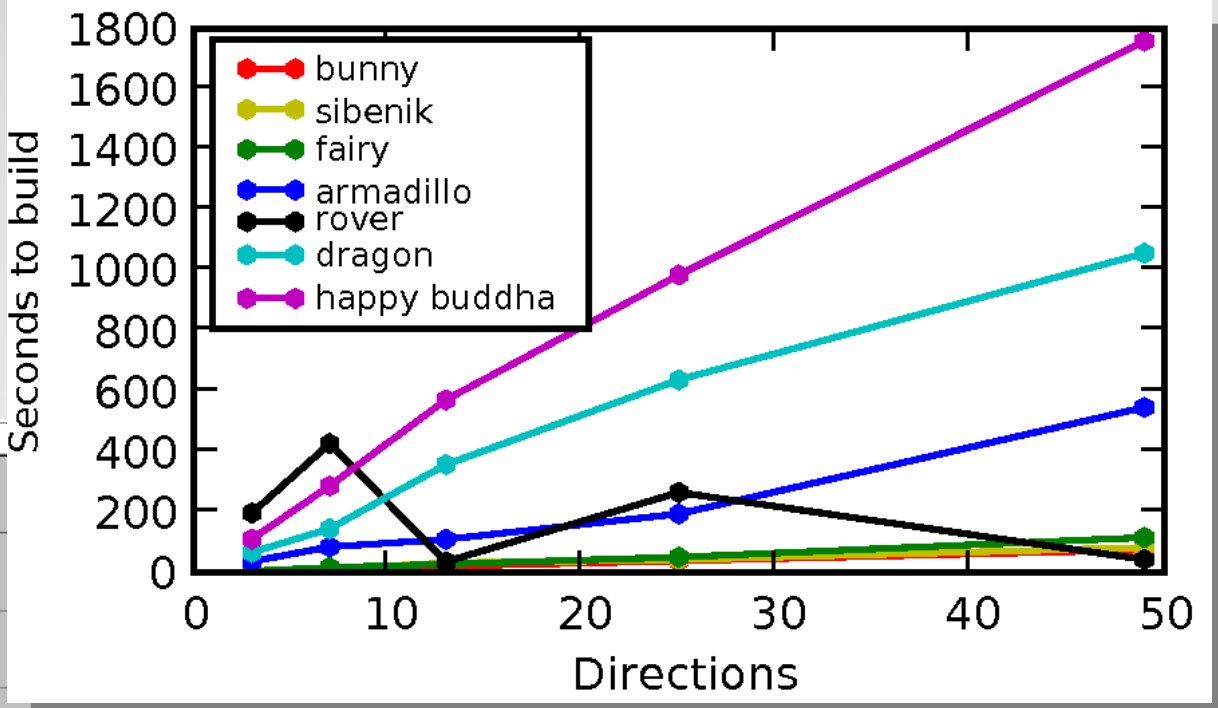
$$Area_{\text{left}}(t_{\text{split}}) = C_{2i}(t_{\text{split}}^2 - t_i^2) + C_{1i}(t_{\text{split}} - t_i) + S_i$$

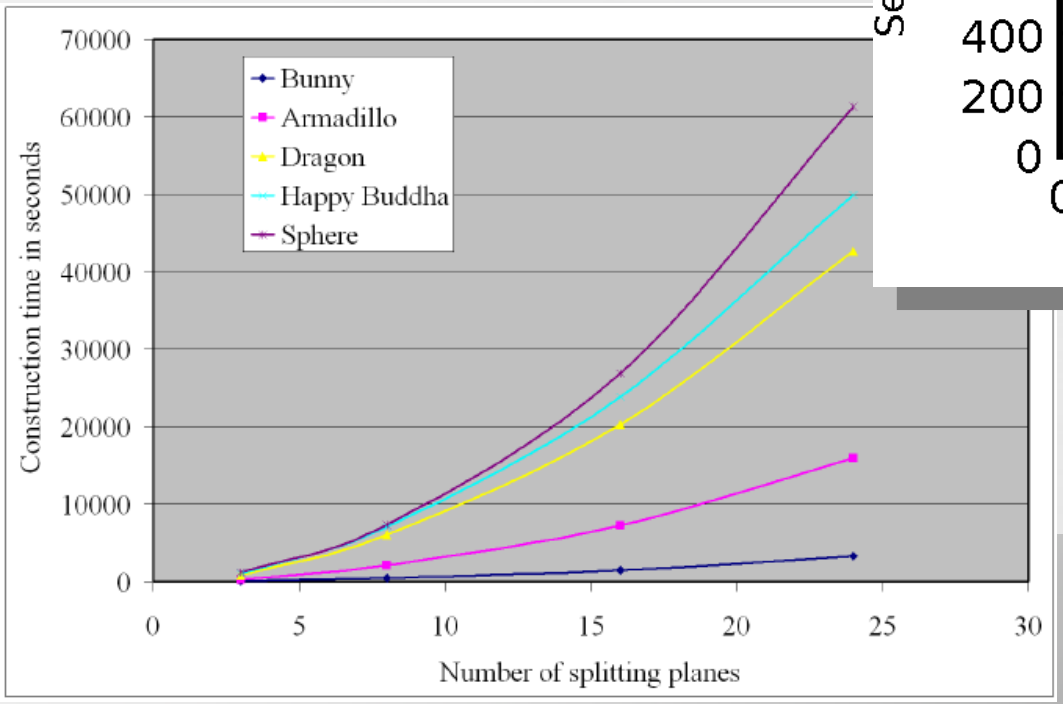
$$Area_{\text{split}}(t_{\text{split}}) = \frac{1}{2}\left(K_{2i}t_{\text{split}}^2 + K_{1i}t_{\text{split}} + K_{0i}\right) \cdot \frac{|D|}{\max(D_x, D_y, D_z)}$$

Build time – Standard Directions



Build time – Golden Ratio Directions





[Kammaje 2007]

Ray tracing the RBSP

- Ray-scene clipping
- Ray preprocessing
- Ray-tree traversal

Ray-scene clipping

- Can clip to scene bounding k-DOP
- AABB = less optimal clipping, but usually faster
- Speeds up tracing
  - Well, only when eye is near or external to scene boundary
  - The more pixels covered, the less clipping helps

Clipping

- Use typical slab intersection for each of the M directions (or for AABB, 3 directions)

Precomputation

- For faster traversal, need projected origin and direction
- Project into each of the M directions

Clipping + Precomp

- The slabs intersection test requires projections!
- Precompute as clipping is performed
- **Almost** worth it
- AABB + SSE precomp is faster

Traversal

```
m = node.getDir();
oproj = node.getSplit() - dot(origin, KdopDirs<M>::d[m]);
dproj = dot(direction, KdopDirs<M>::d[m]);
d = oproj ? oproj / dproj : copysignf(0.5f*lower, dproj);
```

versus

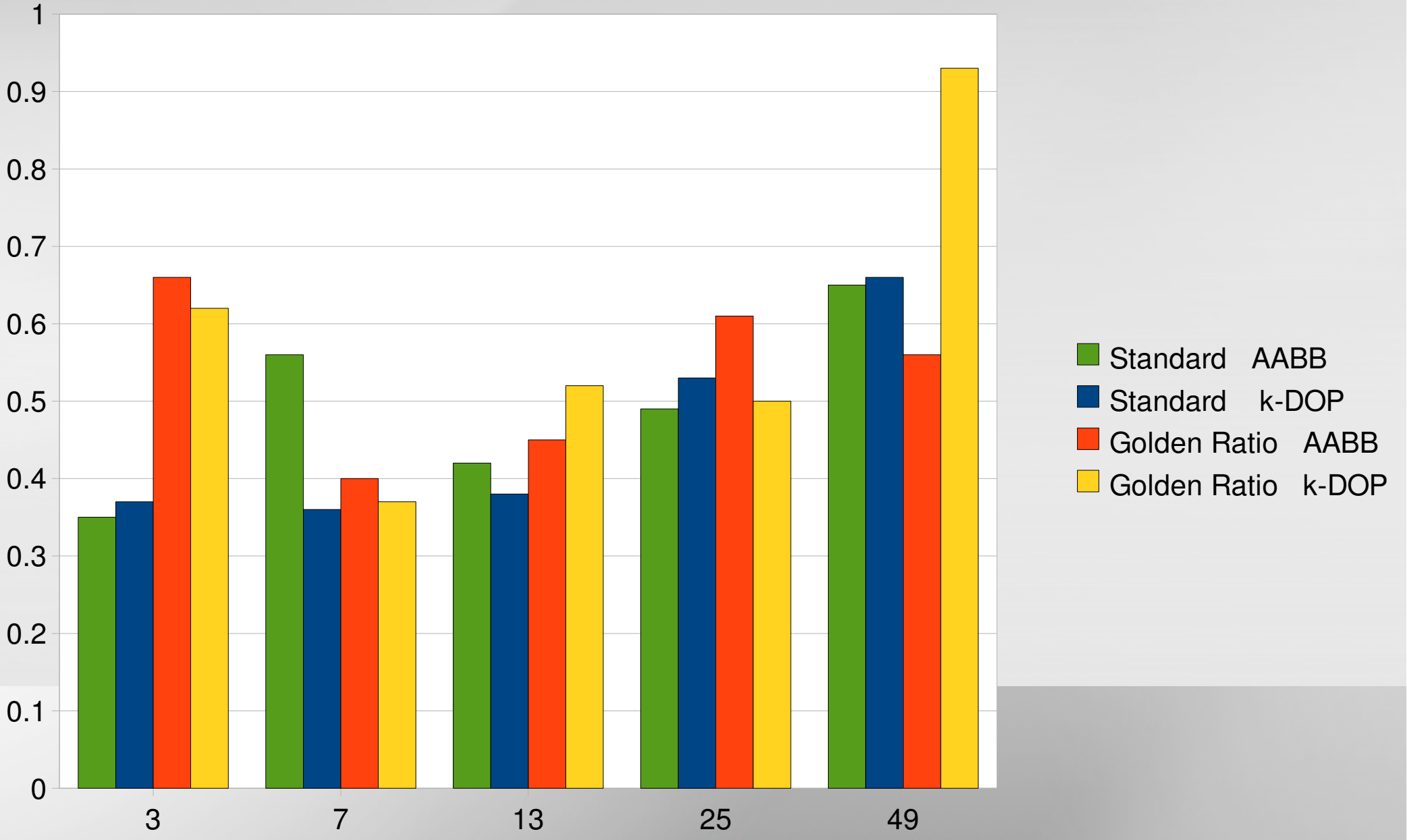
```
m = node.getDir();
oproj = node.getSplit() - orig[m];
d = oproj ? oproj * rcpDir[m] : copysignf(0.5f*lower, rcpDir[m]);
```

results in 5 fewer multiplies, 4 fewer adds, and 1 less division per node traversal

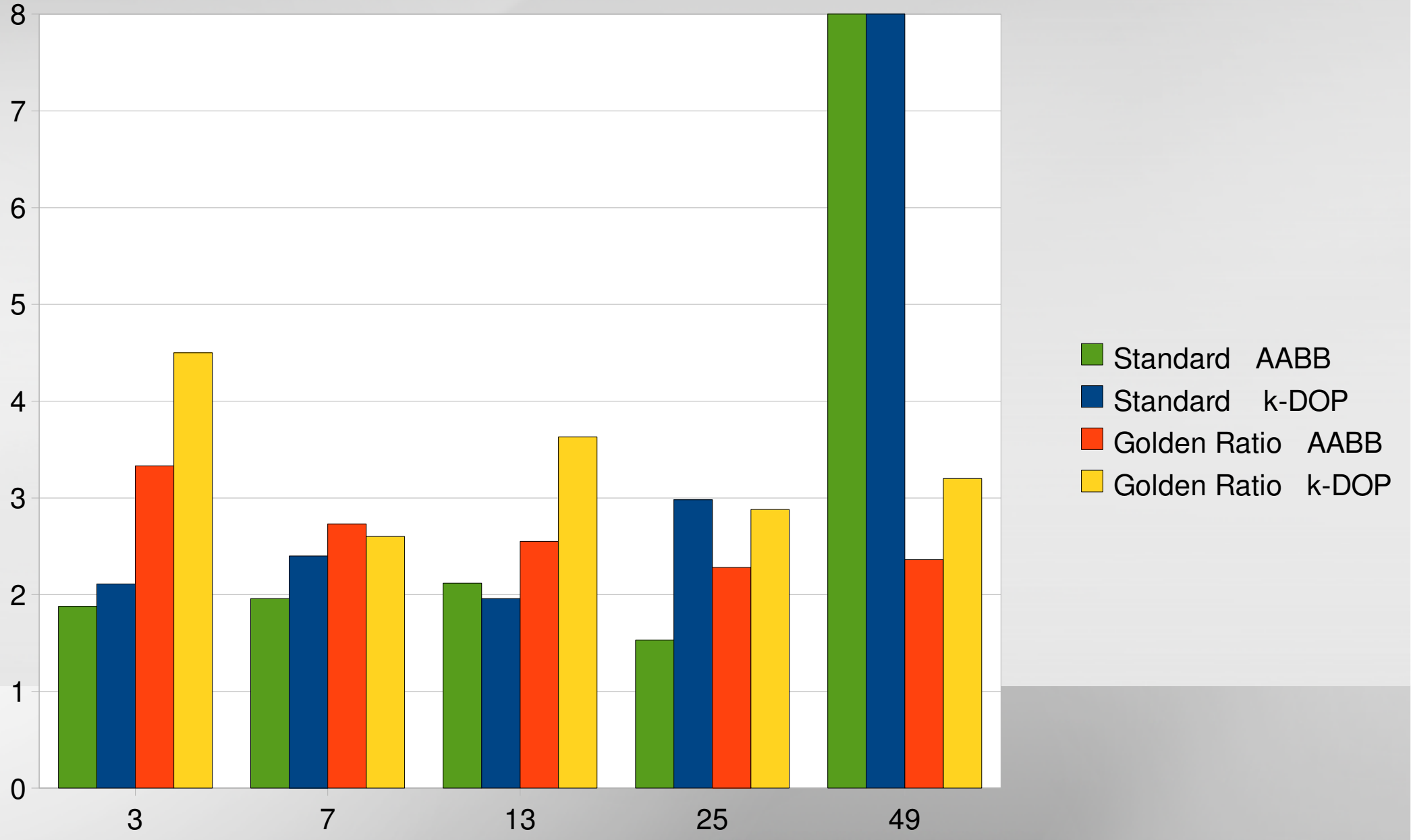
Tradeoffs

- Precomputation not free
  - $2M$ dot products, M divisions
  - As M gets large, precomputation could dominate traversal costs, but at M=49 still slightly faster, even for smallest test scene

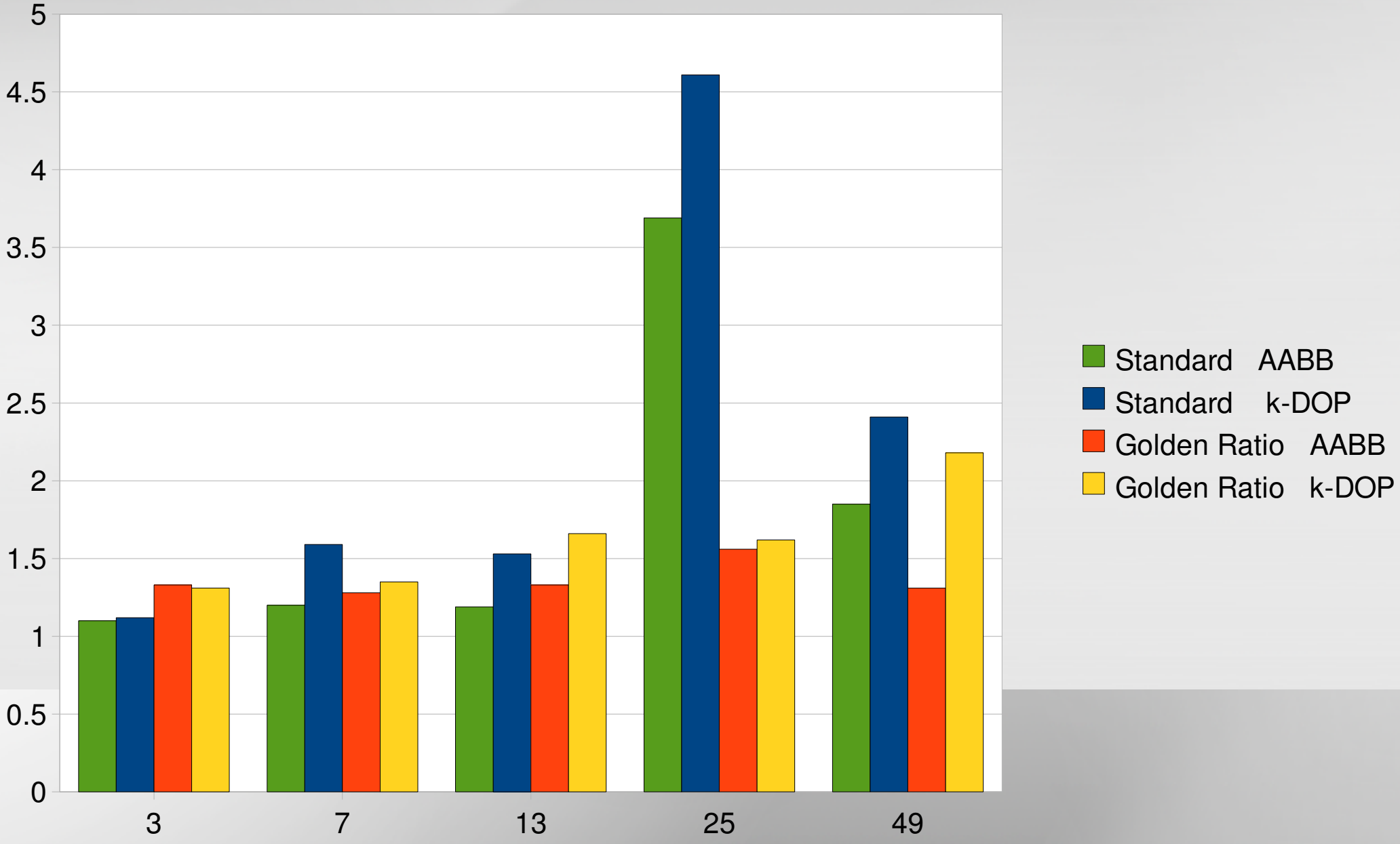
Ray tracing results Bunny



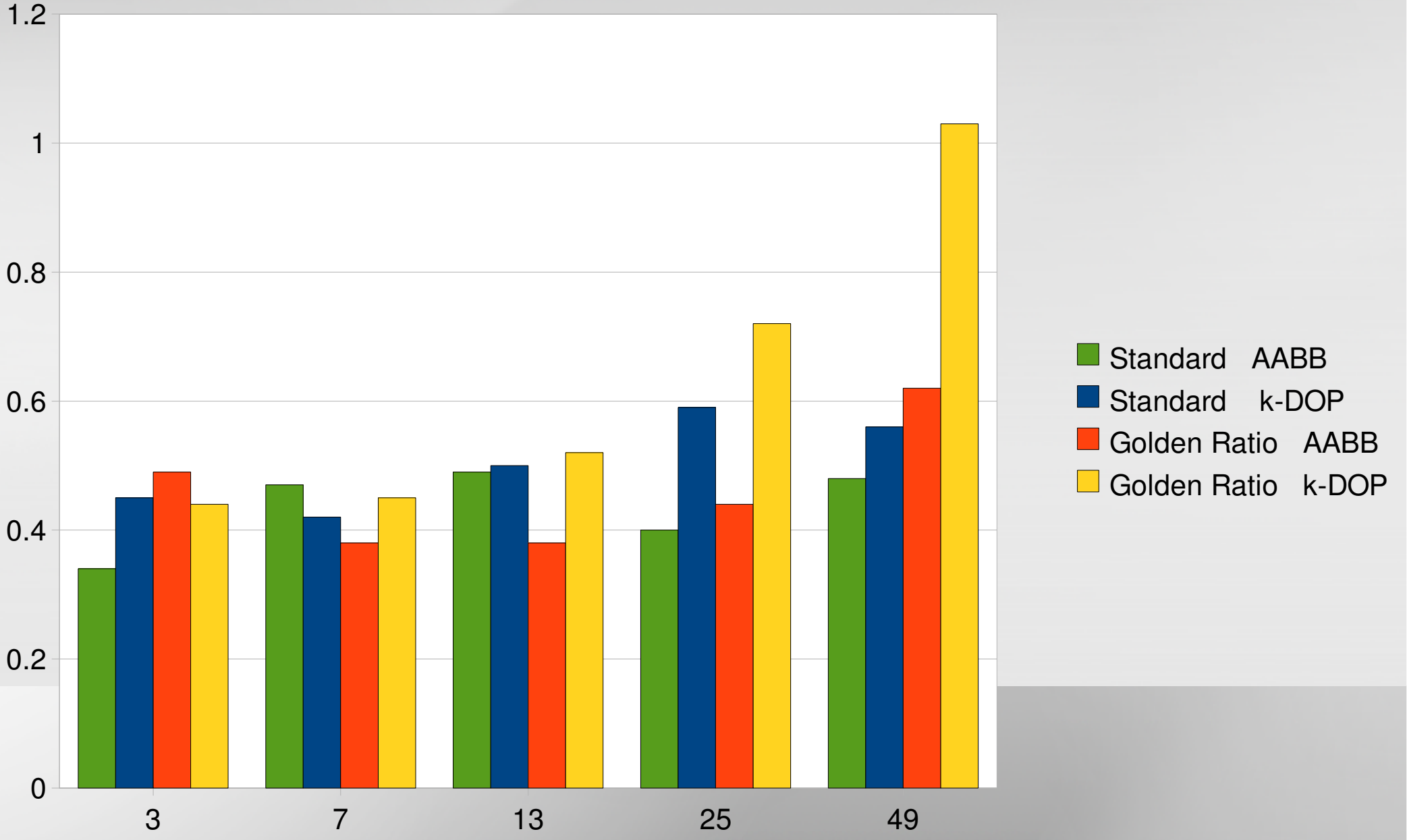
Ray tracing results Sibenik



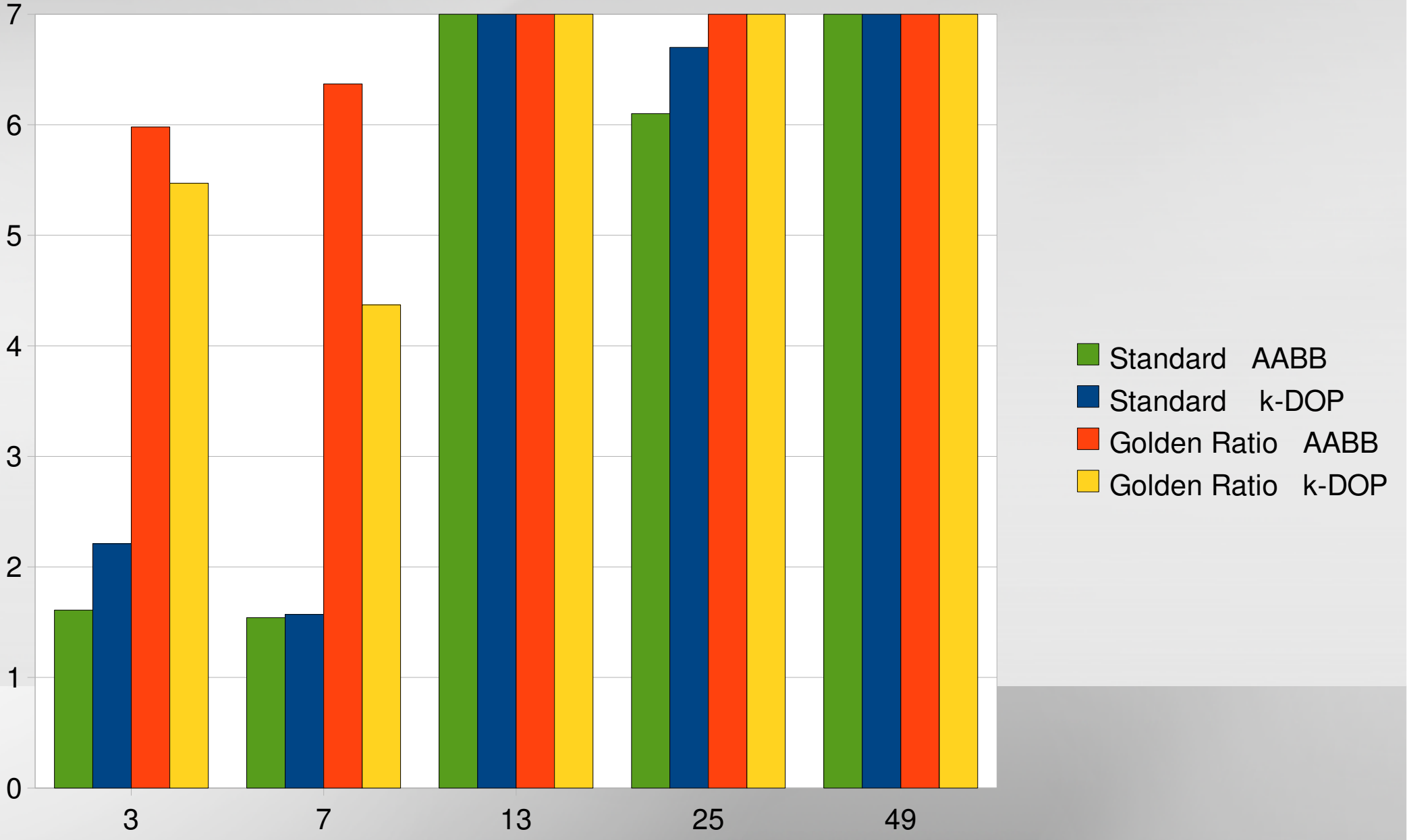
Ray tracing results Fairy



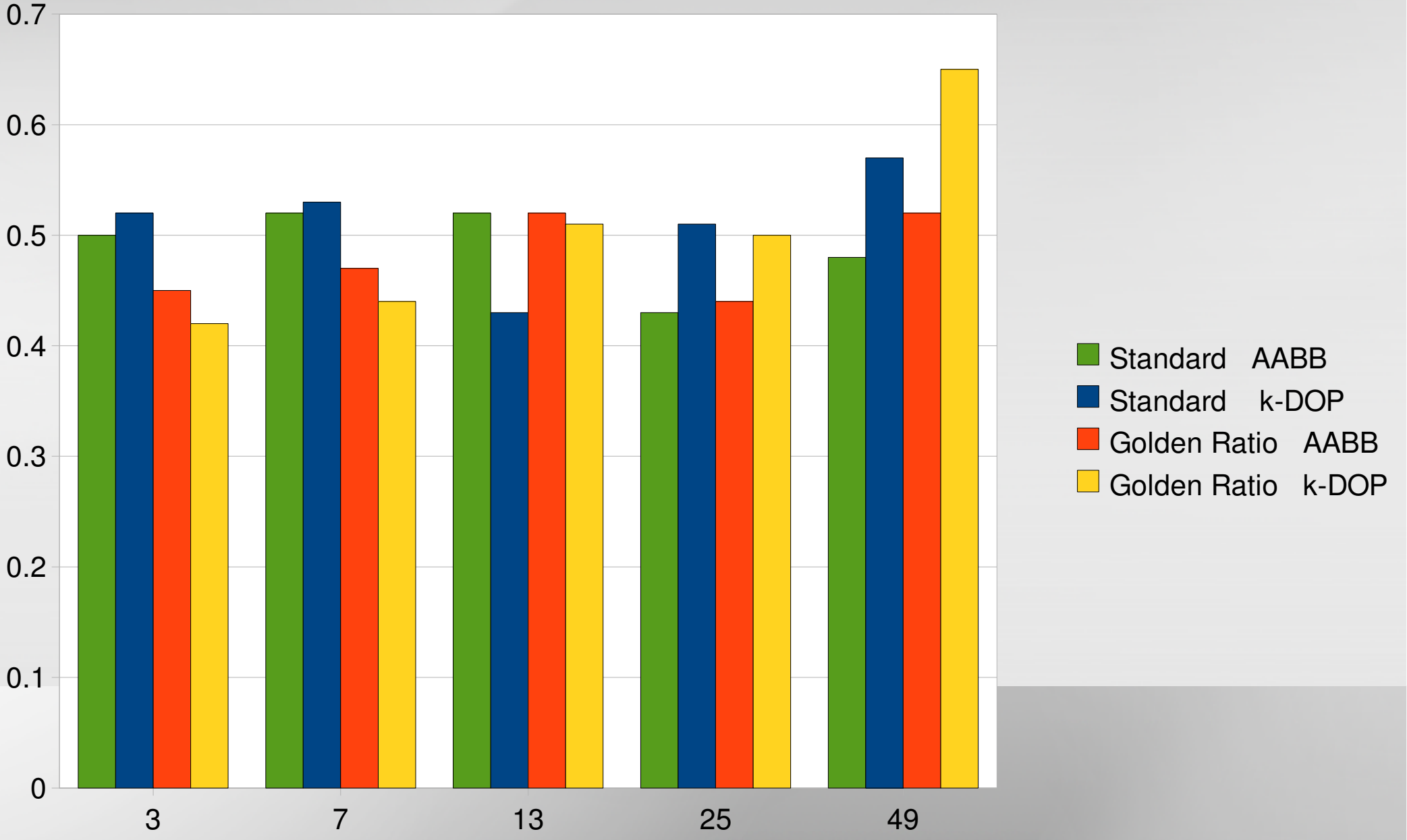
Ray tracing results Armadillo



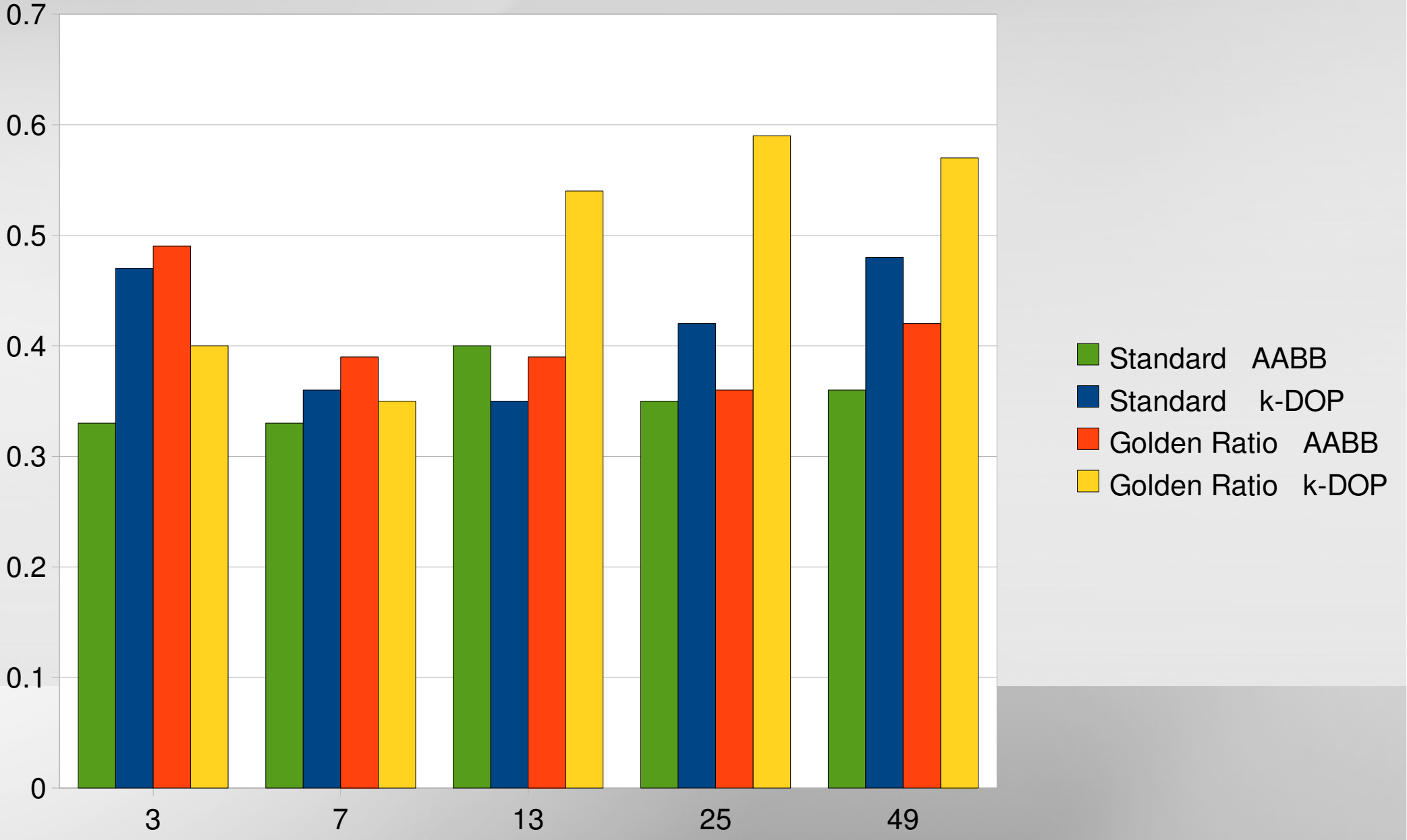
Ray tracing results Rover



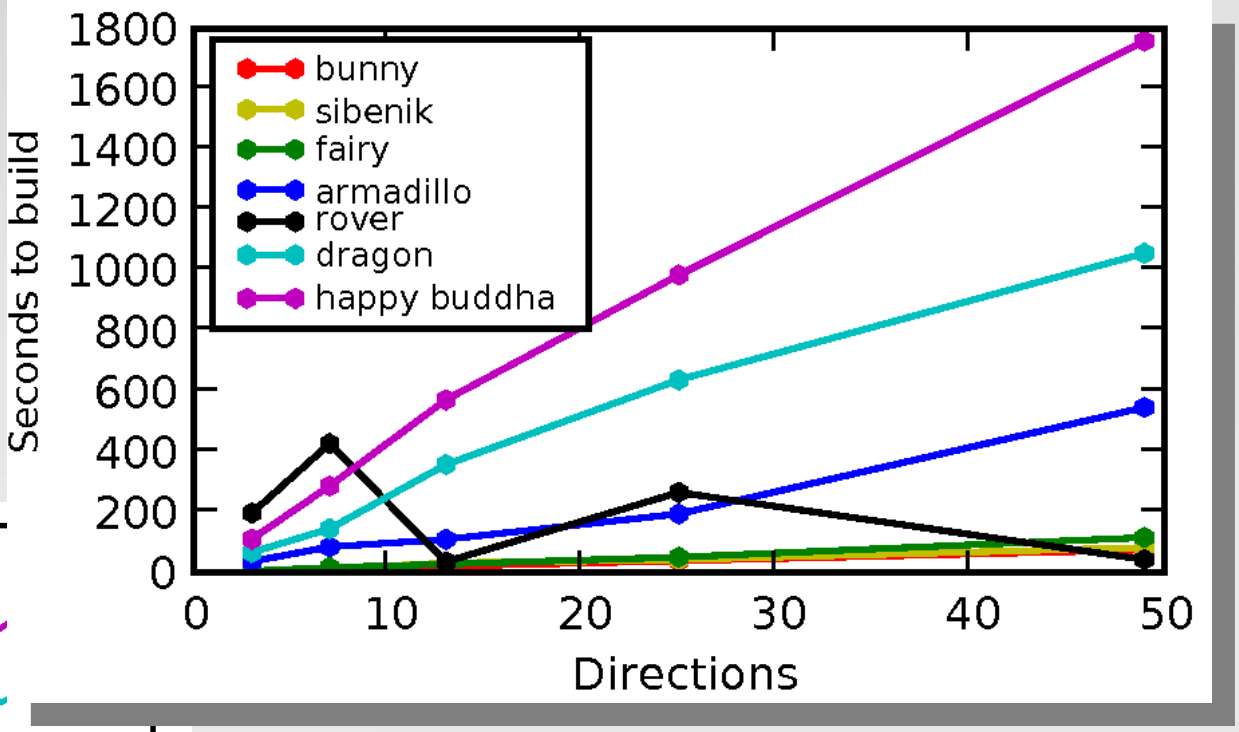
Ray tracing results Dragon

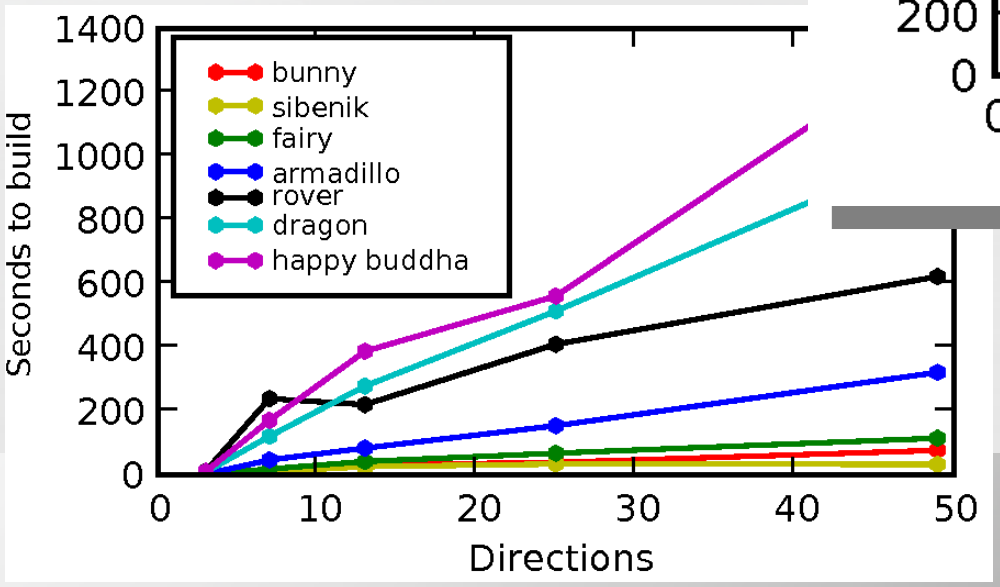


Ray tracing results Buddha



Build times


Seconds to build
Directions
bunny
sibenik
fairy
armadillo
rover
dragon
happy buddha
0
200
400
600
800
1000
1200
1400
1600
1800
0
10
20
30
40
50

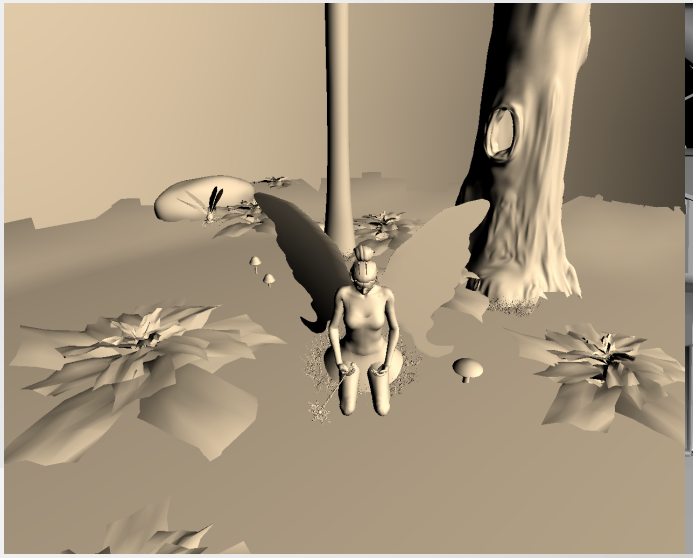
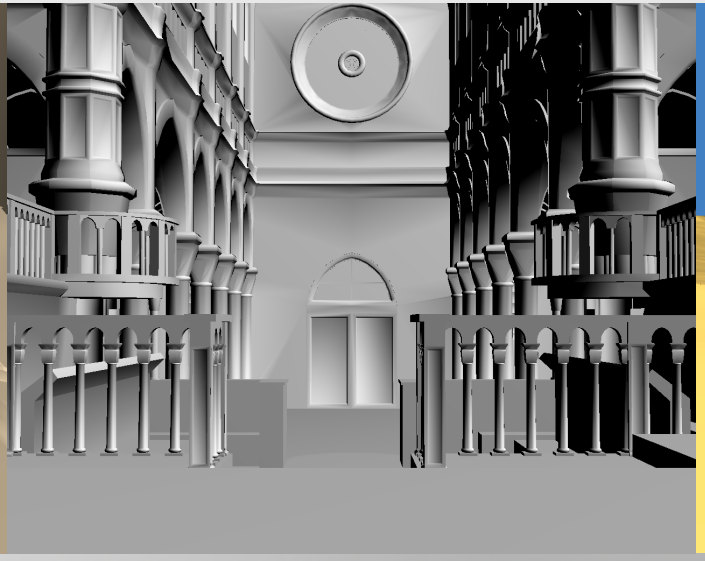
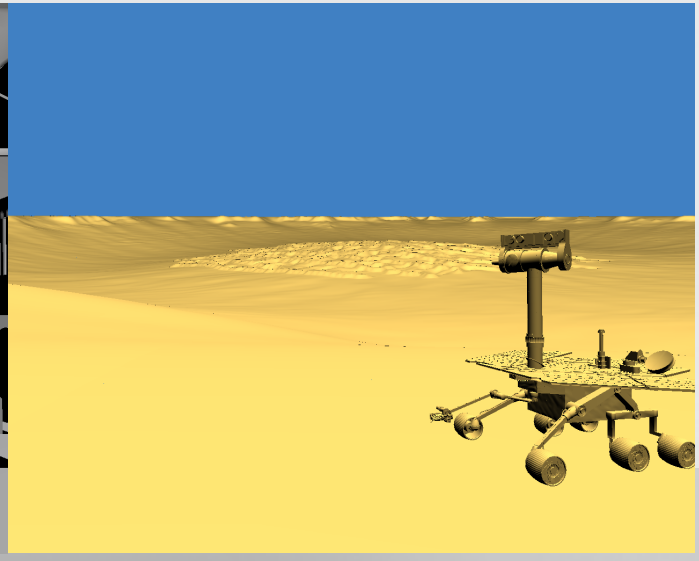

Seconds to build
Directions
bunny
sibenik
fairy
armadillo
rover
dragon
happy buddha
0
200
400
600
800
1000
1200
1400
0
10
20
30
40
50

Conclusions

- New build for RBSP
  - Asymptotically faster $O(M^3 + MN \log N)$
  - ~50x faster build for larger scenes
- Faster render times
  - Roughly 8x
- Number of *k*-DOP faces/edges empirically constant on average, independent of M and standard vs golden ratio

Future Work

- Better RBSP directions
- Combine with *k*d-tree or general BSP
- More robust splitting

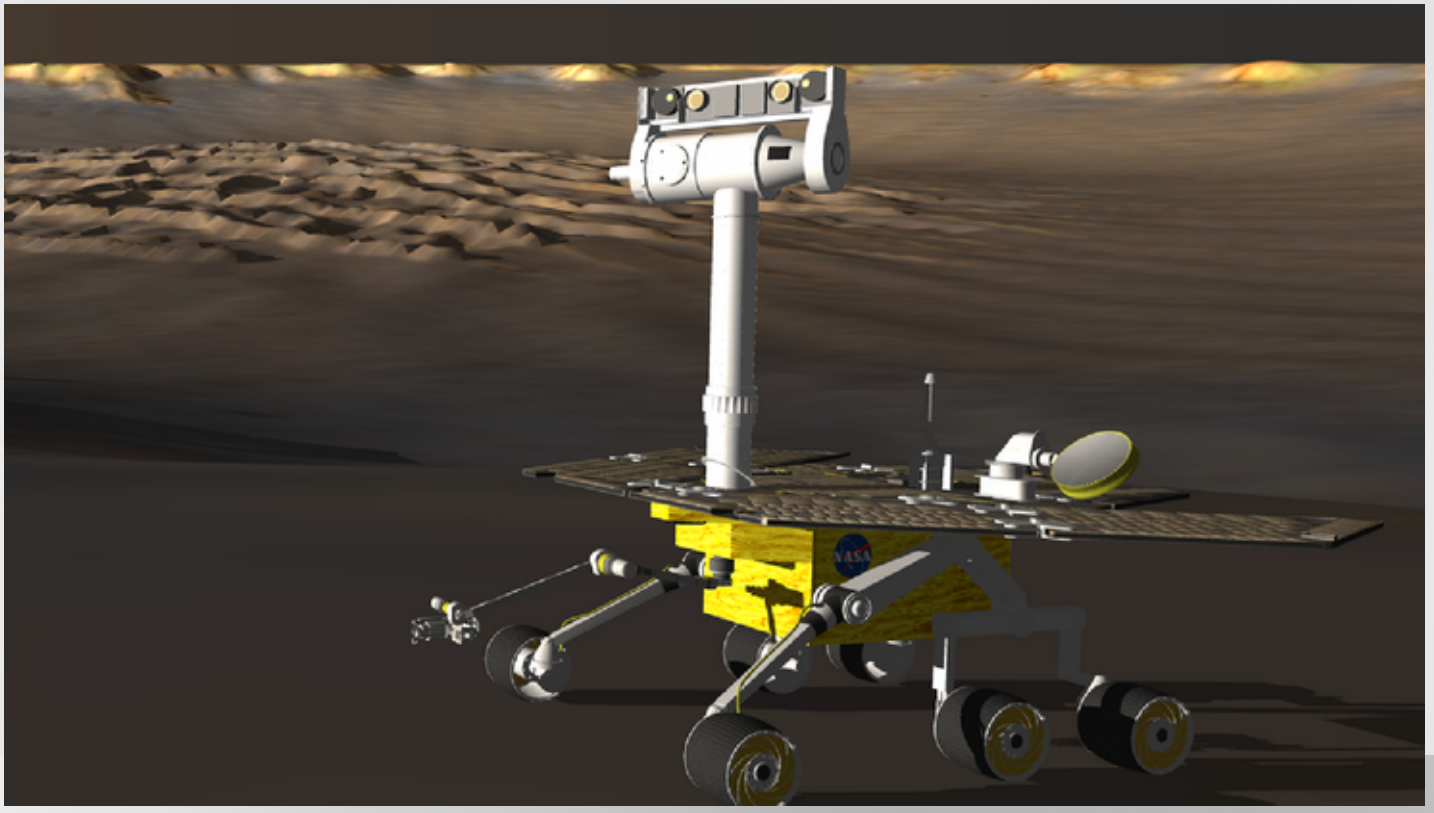




Acknowledgments

- Funding
  - Army PEO-STRI Contract N61339-04-C-0072
  - DoE Contract DE-AC02-05CH11231 (SciDAC)
  - U.S. ED GAANN grant P200A980307
- Thanks to
  - University of Utah, Stanford, and Marko Dabrovic for providing benchmark models
  - Ravi Kammaje and Thiago Ize for some useful discussion

Thanks!

Mars Rover available at
http://cave-wiki.dri.edu/~coming/rover



budge@cs.ucdavis.edu dan.coming@dri.edu