"Space Subdivison for Fast Ray Tracing"

Andrew Glassner

IEEE Computer Graphics & Applications, October 1984,

Volume 4, Number 10, pages 15-22, 1984

A review presentation by

Hasan Gündüz Aydın

Introduction (1)

- The most powerful general synthesis method of the day (1984)
- Realistic images
- However the technique is so slow

Introduction (2)

- Previous work for speeding up the technique (1)
  - Rough rendering of the scene
  - Then investigating those areas of the scene where and additional work seems to be necessary

Introduction (3)

- Previous work for speeding up the technique (2)
  - Parallel computing
  - Each processor handles a subset of the rays

Introduction (4)

- A different approach : decreasing the time required to render a given pixel
- %95 of the total time is spent during the ray-object intersections
- # of intersections is linear wrt to the product of # of ray and # of ojects

Introduction (5)

- Recent work is on the ray-object intersection problem for various classes of objects
- If we want to reduce the time on ray-object intersections
  - Speed up the intersection process itself by specialized hardware
  - Reduce the # of intersections for fully tracing an given ray, approach in this article

Overview of the New Algo. (1)

- Divide the space into voxels (auther uses compartments) and keep a list of the objects in each voxel.
- Find the voxel that the ray intersects
- Check the ray-object intersection just for that voxel
- If hit return the colour, else find the next voxel and repeat the process

Overview of the New Algo. (2)

- The less objects in the voxel, the faster we do the intersection tests
- If finding the next voxel is easy, then we save lots of time wrt naive ray tracing algorithm
- Each voxel has an associated list but not a piece of an object
- List contains those objects whose surfaces pass through the voxel (Show the figure 1 from the article)

Octree Building and Storage (1)

- Fortunately we have octree, so use it.
- Octree is a well-known and understood structure
- Labelling convention (Show the figure 2 from the article) First voxel is 1 (Explained later)
- Dynamic Resolution of the octree scheme: Create a node when we need it

Octree Building and Storage (2)

- **Question**: How to find a previously created voxel?
  - A table where there is an entry for each voxel that has a pointer to the voxel structure (Auther uses node) (Explained later)
    - Vast amount of memory
    - But advantage of extreme speed in finding the address of a voxel structure with a given label (Auther uses name)

Octree Building and Storage (3)

- **Question**: How to find a previously created voxel?
- A large linked list
  - Less memory
  - But more time overhead in finding the address

Octree Building and Storage (4)

- So mix the schemes explained above
  - Hash the label into a small number and then follow a linked list of the voxel structures that have the same label
  - Changing the size of the table is a tradeoff between speed and memory consumption
  - A simple hashing function is voxel label mode the table size

Octree Building and Storage (5)

- Labelling scheme by Gargantini 0..7
- Labelling scheme used here is 1..8
- Because computer can not differentiate between 00 and 0 labels

Octree Building and Storage (6)

- We can find the label of a voxel containing a point (x,y,z) (Show figure 3 from the article)

- **Observation**: When divided a voxel, we create all eight children, when allocating storage ask for a large block then use the first eight for the first and next eight sub-block for the second and so on.

Octree Building and Storage (7)

- **Observation (continue)**: So need to store only the first child in the hash table, others can be indexed using the first child's address (Show the figure 4 from the article)

- Using the above observation, we can reduce the # of entries in the table/linked list structures by a factor of eight

Octree Building and Storage (8)

- Voxel structure
  - Name
  - Subdivision flag
  - Center and size data (Can be omitted and derived on the fly, another time-space tradeoff)
  - An object list pointer

Octree Building and Storage (9)

- While constructing the octree, create new voxels when necessary
- The size of the voxels depends on the programmer's chose (# of objects in the voxel)
- But keep the length of the side of the smallest voxel (Explained later)
- Objects intersections with voxels are done by testing the 6 planes, in the simplest way

Movement to the Next Voxel (1)

- **Fact 1**: No knowledge about how large (or small) the voxels in the space except current one
- **Fact 2**: Movement operation has to be fast
- Idea is to find a point that is quaranteed to be in the next voxel whatever its size
- Basically find the exiting point of the ray from the current voxel

Movement to the Next Voxel (2)

- Then advance the point we find in such a way that we do not exist the next voxel
- We keep the length of the side of the smallest voxel in the space (say minLen), so minLen/2 is a good chose for the advance amount
- Movement operation is designated in figure 7 of the article

Timing and Sample Pictures

- The overall execution time = octree creation time+ image synthesis time
- Multiple views can be generated after octree construction for once for static object database
- Table 1 shows the statistics for old and new ray tracing algorithm

Conclusions

- Naive ray tracing algorithm is famous for its slowness because of the time for ray-object intersections
- This new algorithm cuts the overall time considerably by reducing the ray-object intersections

References

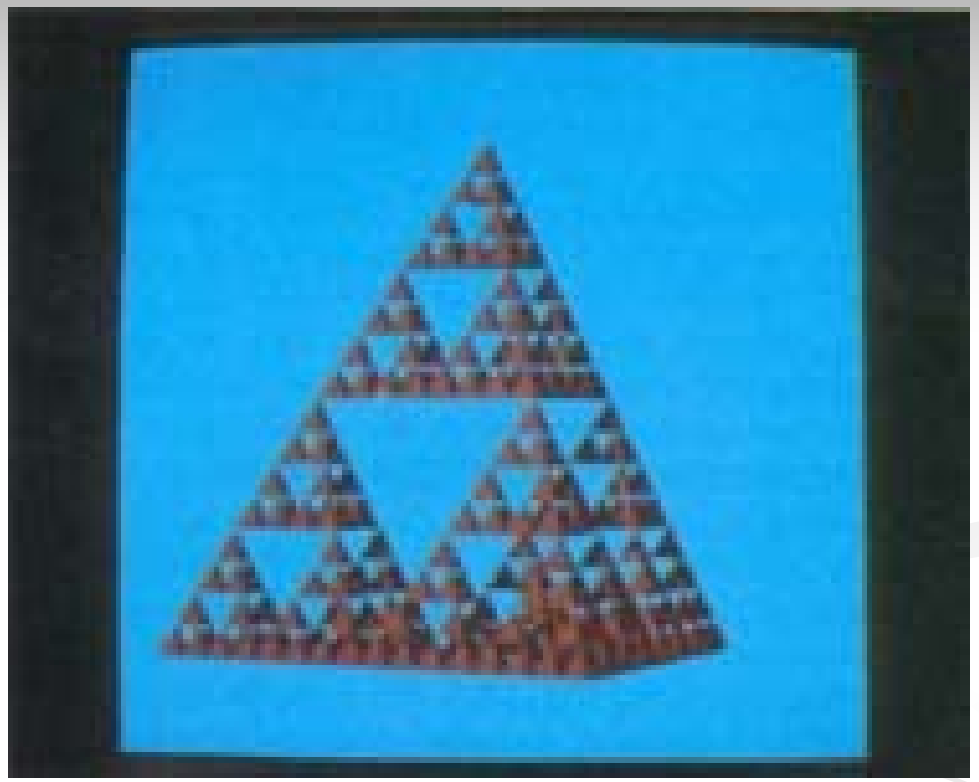
- Glassner, Andrew S., "Space Subdivision for Fast Ray Tracing", IEEE Computer Graphics & Applications, October 1984, Volume 4, Number 10, pages 15-22, 1984 (Can be found at METU Library)

- Auther's web page, http://www.glassner.com/andrew/index.htm

Sample Images (1) (Increasing comlexity)

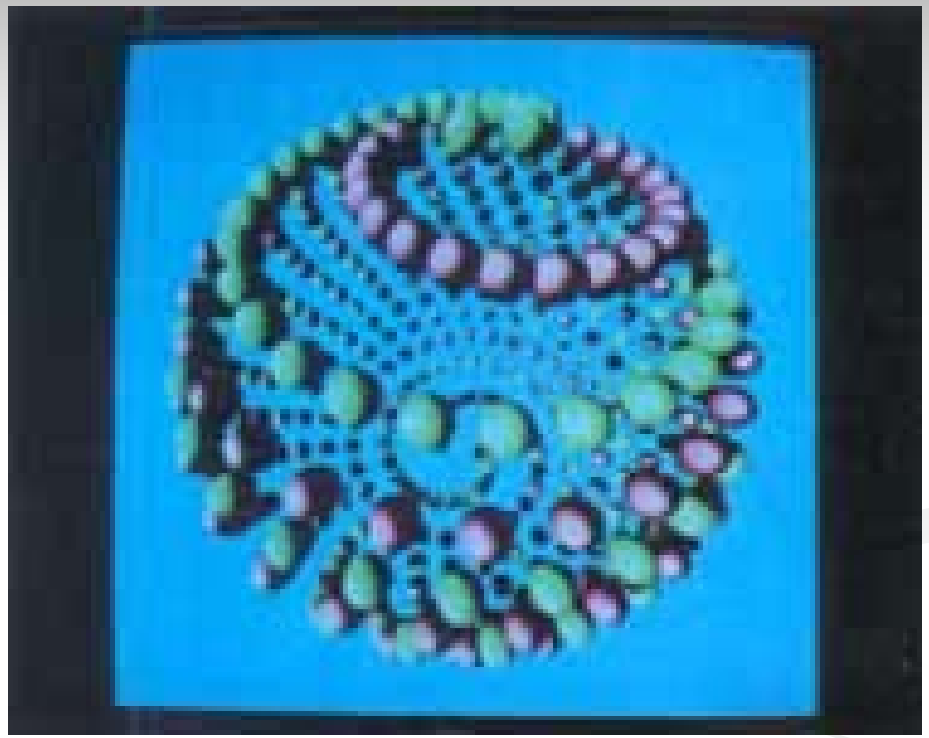
Two perfectly reflecting, intersecting spheres sit between a pair of checkerboards

Sample Images (2) (Increasing comlexity)



A procedurally generated model, similar to a kite designed by Alexander Graham Bell

Sample Images (3) (Increasing comlexity)



Two interweaving spirals of spheres. Note the shadows on the distant balls

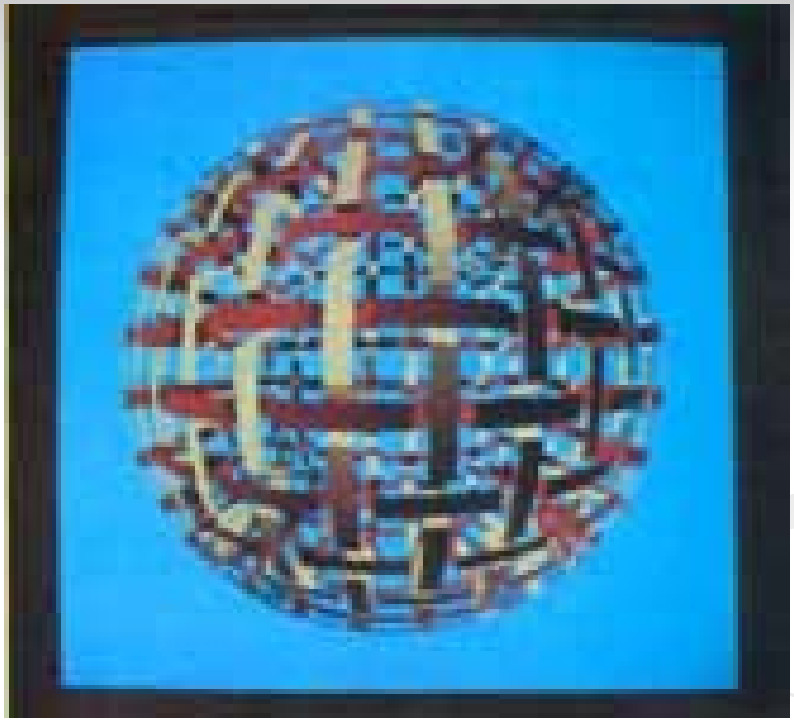
Sample Images (4) (Increasing comlexity)

Two different (4, 4, 3) , tilings of a geodesically projected cube share the surface of a sphere

Sample Images (5) (Increasing comlexity)

A large number of spheres follow the function sin(x)/x for several half-periods

Sample Images (6) (Increasing comlexity)



A single overlap pattern recursively applied to a subdivided cube of frequency 3 and then projected onto a sphere