Real-Time Raytracing

Group 06

Ankit Agarwal - 2011020
Prateek Malhotra - 2011077

Overview

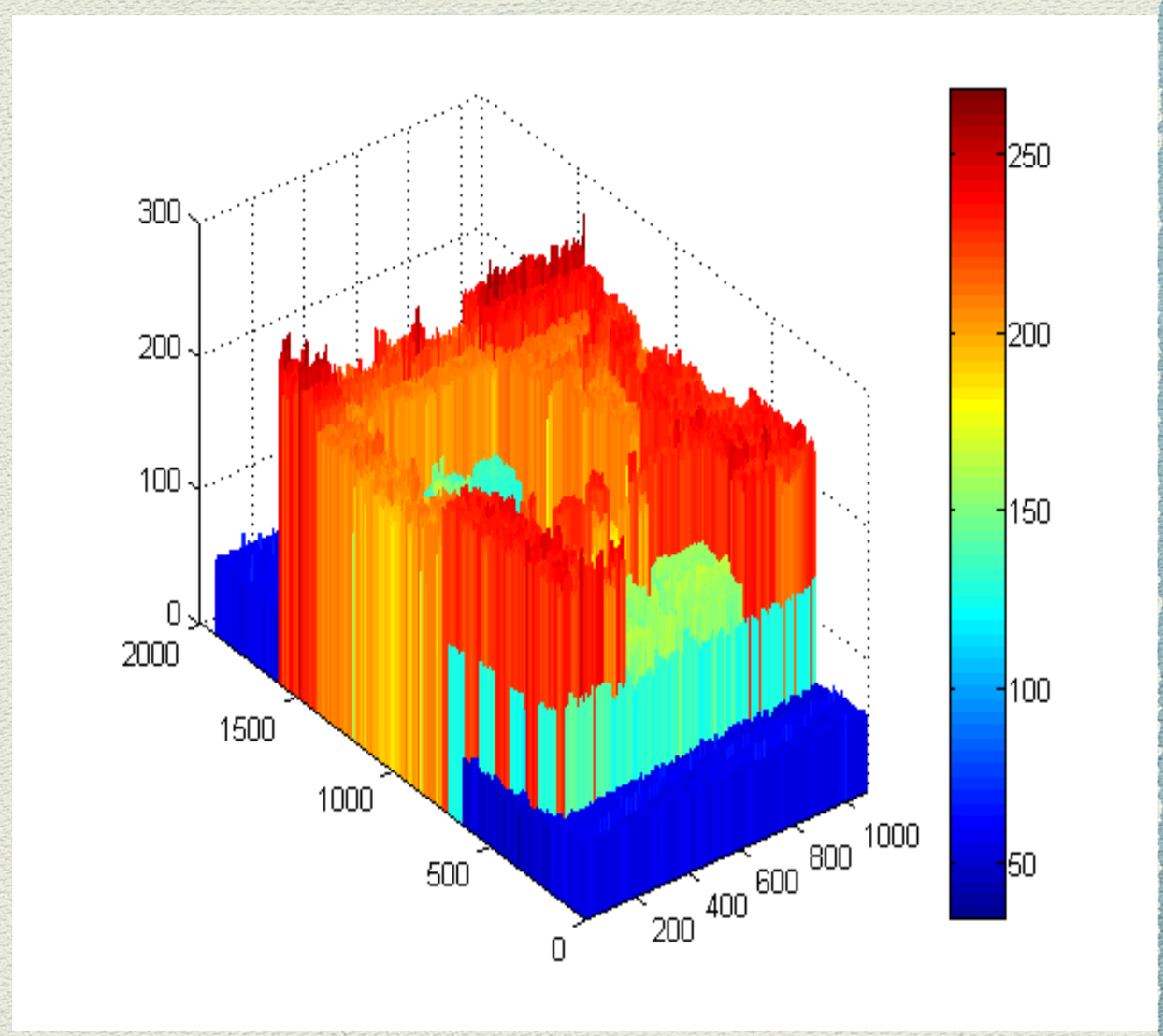
- Summarising previous milestones
- Acceleration Data Structures
- K-d Trees
- Evaluating our K-d Tree implementation
- Performance Gains

CUDA-based Ray Tracer

- Parallel ray-tracing using the CUDA platform
- Phong's Illumination Model + Phong Shading, Shadows, Reflections/Refractions
- Obj file-parser for reading scenes
- Load distribution analysis

Load Distribution Analysis

- Load is not evenly distributed
- Schemes to balance load across threads
- Involve breaking the task into smaller tasks
- Gain is not known to be very high



Acceleration Data Structures

- Data structures to speed up ray-tracing
- Most of the computation is done in intersection tests.
- Involve organising the data so that ray-triangle intersection tests can be minimised.
- Uniform Grids, Octrees, BVHs, K-d Trees

K-d Trees

- Space-partitioning data structure
- A special form of BSP trees
- Space is divided into two halves using a splitting plane
- Splitting planes are axis-aligned
- Results in axis-aligned bounding boxes

K-d Trees - Construction

```
function kdtree (list of points pointList, int depth)
{
    // Select axis based on depth so that axis cycles through all valid values
    var int axis := depth mod k;

    // Sort point list and choose median as pivot element
    select median by axis from pointList;

    // Create node and construct subtrees
    var tree_node node;
    node.location := median;
    node.leftChild := kdtree(points in pointList before median, depth+1);
    node.rightChild := kdtree(points in pointList after median, depth+1);
    return node;
}
```

- K-d Tree Construction for k-d points

K-d Tree Construction - Triangles

```
nodesList :- array containing nodes of the k-d tree.
kdTrianglesList :- array containing all triangles in the scene

constructKdTree(AABB box, list triangles, int dimension)
{
        Initialise this node and add it to the nodesList
        Remember the index at which it has been added.

        Check if leaf criteria met or not.
        if not leaf:
        {
                Sort the triangles on the basis of their centroids in the given dimension.
                Median is the centre of the sorted
                Get the splitPoint value in the given

                Calculate bottom_left and top_right for the children.

                for each triangle in the list:
                        if entire triangle is in one half, find that half and add it to that child.
                else, add the triangle into both children.

                recursively construct left and right subtrees.
          }

        if leaf:
        {
                add triangles into kdTrianglesList
                store start and end index.
        }

    return node.index;
}
```

K-d Tree - Traversal

- Iterative, Stack-based
- Check intersection of box with the ray
- Check if one or both children intersect the ray
- Compute near and far child for stack ordering
- Process nodes at the top of the stack
- For leaves, compute intersection of nodes with the ray
- Empty stack if any triangle intersects with the ray

K-d Trees - Evaluation

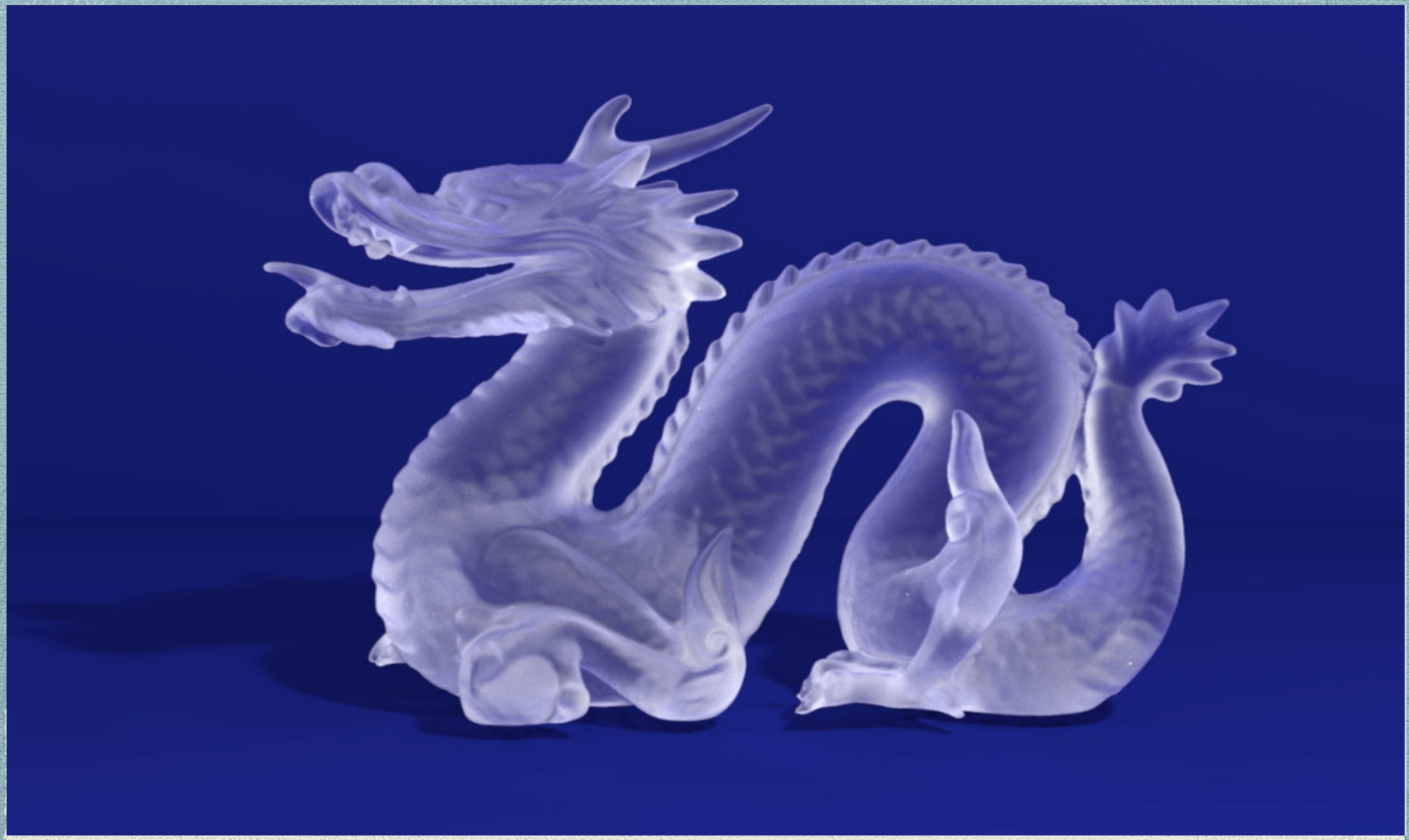
- Pros:
  - Avoid lots of intersection tests
  - AABBs are easy to work with
  - In-order traversal
- Cons:
  - Lead to divergence
  - Memory requests are not coalesced
  - Lots of duplicate triangles
  - Large memory requirements

Some Numbers

All numbers are for the batman scene.

- Without k-d trees:
  - ~17k triangles
  - 17k intersection tests for each thread.
  - Time: 90 secs

- Using k-d trees:
  - ~ 53k triangles (>3x due to duplicates)
  - On average - 1958 intersection tests / thread
    - => 3.7% of total triangles, 11.5% of original numbers
  - Maximum - 8926 intersection tests.
  - Time : < 30 secs => 3x gain



Thank You!