Assignment 2: Bounding Volume Hierarchy

EDAN30

March 30, 2011

This assignment will introduce you to a data structure called the *Bounding Volume Hierarchy* (BVH). In order to pass the assignment you need to complete all tasks. Make sure you can explain your solutions in detail.

1 Scene setup

You will need to create two new files – `bvhaccelerator.cpp` and `bvhaccelerator.h` containing the BVH implementation and declaration. The BVH class should inherit from the `RayAccelerator`-class.

2 BVH Construction

First we need to build the BVH from a list of incoming `Itersectables`. Starting with the entire set of `Itersectables`, we must (somewhat intelligently) split that in to two sub-sets. This "simple" process is repeated recursively until there are only a few primitives per set. Each sub-set has its own bounding box which is, at most, as big as the bounding box of the parent set. In the end we will have a binary tree with an approximate $O(\log n)$ search complexity.

Task 1: *Start by inserting the "buildSimple" scene to your project.* You will find the code in appendix B.1. Make sure that the `scramble`-parameter is set to *false.*

Task 2: *Implement a BVH skeleton and replace the list accelerator object with your BVH in the beginning of the `main`-function.* The project should now compile, but it will not run properly.

Task 3: *Implement the build()-function, as described in the seminar. Only* *implement the base case (no recursion yet).* Make sure that your world bounding box is:

```
World Bounds:
Min:      -40.5     -40.5     -40.5
Max:       40.5      40.5      40.5
```

Now it's time to actually do the construction. So, what is a sensible splitting criteria? *Mid-point splitting* is easy to understand and implement, so let's start with that.

Task 4: *Implement the recursion portion of the BVH construction routine. Use mid-point splitting.* Without being able to intersect anything it is difficult to debug, however. Therefore we also *include the debug printing function in your BVH.* You find the code for the print-function in appendix A. When you run your build-function, your output should match the output listed in appendix B.2.

Task 5: *Change the parameter* `scramble` *of* `buildSimple` *to* true*.* The output of the print-function should now (approximately) match that of appendix B.3.

When you are confident that your BVH construction routine is working properly, *remove the print function call* (or the console will be flooded later), and move on to the next set of assignments.

3 BVH Traversal

Next we need to implement intersection testing, both a closest hit- and a boolean-test. The seminar slides should give you enough background to understand how this is done.

We will first implement the closest hit function, and ignore shadow rays for now (they need boolean intersection tests). Just make the boolean-test always return *false*, and your shadows are effectively disabled.

Task 6: *Implement a closest hit intersection test.* You should try to keep the implementation as simple as possible to begin with. Add optimizations when you are sure that the intersection code is working. Remember that *premature optimization is the root of all evil and may implode the Universe.*

Task 7: *Switch to the "buildElephants" scene.* The code can be found in appendix C. Your image should roughly match the ones in the seminar, minus the shadows.

Task 8: *Implement a boolean intersection test.* Remember that the boolean intersection code will be similar to the closest hit code, so that's a good starting point.

4 Surface Area Heuristic (SAH) (Optional)

By building a good tree, i.e. putting more thought in to the splitting criteria, we can gain some more performance.

Optional 1: *Alter the construction function to use SAH instead of midpoint splitting.*

5 Optimization (Optional)

Finally, there is an array of small optimizations that can be implemented. The construction function usually constitutes a small part of the execution time, so it's probably a better idea to concentrate most of the optimization efforts on the traversal code. Implementing a better tree construction technique (better splitting criteria) is of course also a good area of improvement. For more tips, check the seminar slides.

Optional 2: *Try to optimize the BVH intersection and construction code as much as possible.*

6 Conclusion

In this assignment we have implemented a BVH which has improved rendering times tremendously. Cranking up the number of rays per second is impervious in rendering realistic scenes.

A BVH Debug Printing Function

Use the following print function to make sure that you have built a good tree. Copy and paste this code into `bvhaccelerator.cpp`. (You will need to put declarations in `bvhaccelerator.h` as well of course.)

```
#include <iomanip>
void BVHAccelerator::print_rec(BVHNode *node, int depth)
{
  std::cout << std::setw(depth * 2) << ' ';
  if (node->isLeaf())
    std::cout << "Leaf<Primitives: " << node->getNObjs() << ", First primitive: " << node->getIndex() << ">" << std::endl;
  else {
    std::cout << "Node<Primitives: " << node->getNObjs() << ">" << std::endl;
    BVHNode *left = &nodeList[node->getIndex()];
    BVHNode *right = &nodeList[node->getIndex() + 1];
    print_rec(left, depth + 1);
    print_rec(right, depth + 1);
  }
}

void BVHAccelerator::print()
{
  std::cout << "Printing nodes..." << std::endl;
  std::cout << "World Bounds: " << std::endl;
  std::cout << "Min: " << root->getAABB().mMin;
  std::cout << "Max: " << root->getAABB().mMax << std::endl;
  print_rec(root, 1);
}
```

B Simple Test Scene

Use this function to verify that your BVH construction function works as expected.

B.1 Scene Construction Function

Copy and paste this code into `main.cpp`

```
static void buildSimple(Scene& scene, bool scrambled)
{
  PointLight *pointLight0 = new PointLight(Point3D(-20.0f, 20.0f, 60.0f), Color(1.5f, 1.5f, 1.5f));
  scene.add(pointLight0);

  // Setup material
  Diffuse *material = new Diffuse(Color(0.0f,0.2f,1.5f));
  material->setReflectivity(0.4f);

  // Add spheres
  for (int i = 0; i < 80; i++) {
    float x, y, z;
    if (scrambled) {
      x = -39.5f + (float) ((i * 13) % 80);
      y = -39.5f + (float) ((i * 7) % 80);
      z = -39.5f + (float) ((i * 29) % 80);
    } else {
      x = -39.5f + (float) (i % 80);
      y = -39.5f + (float) (i % 80);
      z = -39.5f + (float) (i % 80);
    }
    Sphere* sphere = new Sphere(1.0, material);
    sphere->setTranslation(Vector3D(x, y, z));
    scene.add(sphere);
  }
}
```

You might also want to change your camera to match the seminar images:

```
  Camera* camera = new Camera(&output);
  Point3D pos(0.0f, 0.0f, 116.0f);
  Point3D target(0.0f, 0.0f, 0.0f);
  Vector3D up(0.0f, 1.0f, 0.0f);
  camera->setLookAt(pos, target, up, 58.0f);
```

B.2 Expected Result (Non-scrambled positions)

You should get the same (or a similar) result if you print your contructed tree using the print-function, with the `scrambled` parameter set to *false*.

```
World Bounds:
Min:      -40.5     -40.5     -40.5
Max:       40.5      40.5      40.5

  Node<Primitives: 80>
    Node<Primitives: 40>
      Node<Primitives: 20>
        Node<Primitives: 10>
          Node<Primitives: 5>
            Leaf<Primitives: 3, First primitive: 0>
            Leaf<Primitives: 2, First primitive: 3>
          Node<Primitives: 5>
            Leaf<Primitives: 3, First primitive: 5>
            Leaf<Primitives: 2, First primitive: 8>
        Node<Primitives: 10>
          Node<Primitives: 5>
            Leaf<Primitives: 3, First primitive: 10>
            Leaf<Primitives: 2, First primitive: 13>
          Node<Primitives: 5>
            Leaf<Primitives: 3, First primitive: 15>
            Leaf<Primitives: 2, First primitive: 18>
      Node<Primitives: 20>
        Node<Primitives: 10>
          Node<Primitives: 5>
            Leaf<Primitives: 3, First primitive: 20>
            Leaf<Primitives: 2, First primitive: 23>
          Node<Primitives: 5>
            Leaf<Primitives: 3, First primitive: 25>
            Leaf<Primitives: 2, First primitive: 28>
        Node<Primitives: 10>
          Node<Primitives: 5>
            Leaf<Primitives: 3, First primitive: 30>
            Leaf<Primitives: 2, First primitive: 33>
          Node<Primitives: 5>
            Leaf<Primitives: 3, First primitive: 35>
            Leaf<Primitives: 2, First primitive: 38>
    Node<Primitives: 40>
      Node<Primitives: 20>
        Node<Primitives: 10>
          Node<Primitives: 5>
            Leaf<Primitives: 3, First primitive: 40>
            Leaf<Primitives: 2, First primitive: 43>
          Node<Primitives: 5>
            Leaf<Primitives: 3, First primitive: 45>
            Leaf<Primitives: 2, First primitive: 48>
        Node<Primitives: 10>
          Node<Primitives: 5>
            Leaf<Primitives: 3, First primitive: 50>
            Leaf<Primitives: 2, First primitive: 53>
          Node<Primitives: 5>
            Leaf<Primitives: 3, First primitive: 55>
            Leaf<Primitives: 2, First primitive: 58>
      Node<Primitives: 20>
        Node<Primitives: 10>
          Node<Primitives: 5>
            Leaf<Primitives: 3, First primitive: 60>
            Leaf<Primitives: 2, First primitive: 63>
          Node<Primitives: 5>
            Leaf<Primitives: 3, First primitive: 65>
            Leaf<Primitives: 2, First primitive: 68>
        Node<Primitives: 10>
          Node<Primitives: 5>
            Leaf<Primitives: 3, First primitive: 70>
            Leaf<Primitives: 2, First primitive: 73>
          Node<Primitives: 5>
            Leaf<Primitives: 3, First primitive: 75>
            Leaf<Primitives: 2, First primitive: 78>
```

B.3 Expected Result (Scrambled positions)

You should get the same (or a similar) result if you print your contructed tree using the print-function, with the `scrambled` parameter set to *true*.

```
World Bounds:
Min:      -40.5     -40.5     -40.5
Max:       40.5      40.5      40.5

  Node<Primitives: 80>
    Node<Primitives: 40>
      Node<Primitives: 21>
        Node<Primitives: 8>
          Leaf<Primitives: 4, First primitive: 0>
          Leaf<Primitives: 4, First primitive: 4>
        Node<Primitives: 13>
          Node<Primitives: 7>
            Leaf<Primitives: 4, First primitive: 8>
            Leaf<Primitives: 3, First primitive: 12>
          Node<Primitives: 6>
            Leaf<Primitives: 3, First primitive: 15>
            Leaf<Primitives: 3, First primitive: 18>
      Node<Primitives: 19>
        Node<Primitives: 7>
          Leaf<Primitives: 4, First primitive: 21>
          Leaf<Primitives: 3, First primitive: 25>
        Node<Primitives: 12>
          Node<Primitives: 6>
            Leaf<Primitives: 3, First primitive: 28>
            Leaf<Primitives: 3, First primitive: 31>
          Node<Primitives: 6>
            Leaf<Primitives: 3, First primitive: 34>
            Leaf<Primitives: 3, First primitive: 37>
    Node<Primitives: 40>
      Node<Primitives: 19>
        Node<Primitives: 6>
          Leaf<Primitives: 3, First primitive: 40>
          Leaf<Primitives: 3, First primitive: 43>
        Node<Primitives: 13>
          Node<Primitives: 6>
            Leaf<Primitives: 4, First primitive: 46>
            Leaf<Primitives: 2, First primitive: 50>
          Node<Primitives: 7>
            Leaf<Primitives: 4, First primitive: 52>
            Leaf<Primitives: 3, First primitive: 56>
      Node<Primitives: 21>
        Node<Primitives: 6>
          Leaf<Primitives: 3, First primitive: 59>
          Leaf<Primitives: 3, First primitive: 62>
        Node<Primitives: 15>
          Node<Primitives: 8>
            Leaf<Primitives: 4, First primitive: 65>
            Leaf<Primitives: 4, First primitive: 69>
          Node<Primitives: 7>
            Leaf<Primitives: 4, First primitive: 73>
            Leaf<Primitives: 3, First primitive: 77>
```

C Elephant Scene

A scene consisting of two elephants and a plane. Copy and paste this code into `main.cpp`, for example.

```
static void buildElephant(Scene& scene)
{
  PointLight *pointLight0 = new PointLight(Point3D(-50.0f, 60.0f, 20.0f), Color(0.4f, 0.7f, 0.7f));
  scene.add(pointLight0);

  PointLight *pointLight1 = new PointLight(Point3D(70.0f, 140.0f, -7.0f), Color(2.5f, 2.5f, 2.5f));
  scene.add(pointLight1);

  Diffuse *planeMaterial = new Diffuse(Color(1.0f,1.0f,1.0f));
  planeMaterial->setReflectivity(0.75f);
  Mesh* plane = new Mesh("data/plane.obj", planeMaterial);
  plane->setScale(20.0f);
  plane->setTranslation(Vector3D(0, -10, 0));
  scene.add(plane);

  Diffuse *elephantMaterial0 = new Diffuse(Color(0.4f,0.7f,1.0f));
  elephantMaterial0->setReflectivity(0.55f);
  Mesh* elephant0 = new Mesh("data/elephant.obj", elephantMaterial0);
  elephant0->setScale(1.1f);
  elephant0->setRotation(0.0f, 220.0f, 0.0f);
  elephant0->setTranslation(Vector3D(-12, -10, 1));
  scene.add(elephant0);

  Diffuse *elephantMaterial1 = new Diffuse(Color(0.4f,1.0f,0.7f));
  elephantMaterial1->setReflectivity(0.20f);
  elephantMaterial1->setTransparency(0.50f);
  elephantMaterial1->setIndexOfRefraction(1.1f);
  Mesh* elephant1 = new Mesh("data/elephant.obj", elephantMaterial1);
  elephant1->setScale(1.35f);
  elephant1->setRotation(0.0f, 190.0f, 0.0f);
  elephant1->setTranslation(Vector3D(8.0f, -10, -5));
  scene.add(elephant1);
}
```

Also change the camera parameters found in the `main`-function to:

```
  Camera* camera = new Camera(&output);
  Point3D pos(27.0f, 13.0f, 21.0f);
  Point3D target(0.0f, -4.0f, 0.0f);
  Vector3D up(0.0f, 1.0f, 0.0f);
  camera->setLookAt(pos, target, up, 52.0f);
```