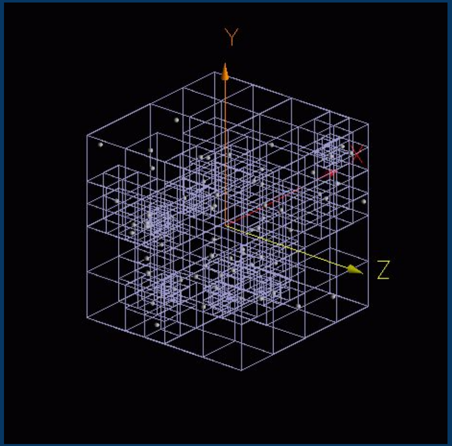
Barnes Hut API



Sawyer Mifsud, Robert Schwartz,
Odysseas Fourakis, Cristian Tharin

Space

Space is vast, full of objects such as: planets, comets, meteors, etc. We call these objects "bodies."

Bodies in Space

There are billions upon billions of bodies in space, all interacting with each others gravitational forces.

These gravitational forces depend on the objects’ masses and positions relative to each other.

Calculating these forces by hand would be tedious and time consuming.

The Barnes Hut Algorithm

The Barnes Hut Algorithm is much quicker and easier alternative for these calculations.

The bodies are treated as a cluster, and calculations are done on the center of gravity of the 'cluster', rather than each individual body.

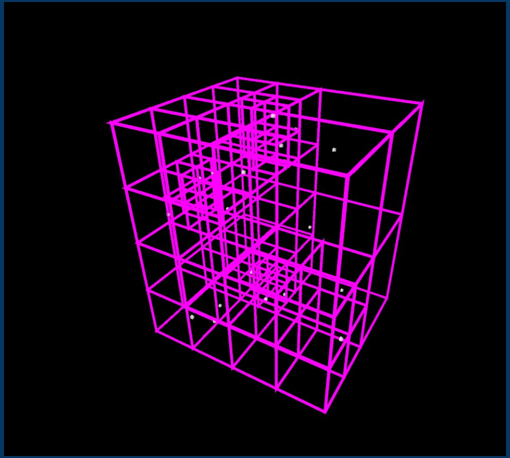
How does it work?

Physical space is divided into eight sub-spaces called octants.

We then further subdivide these octants into more octants until we reach a preset limit.

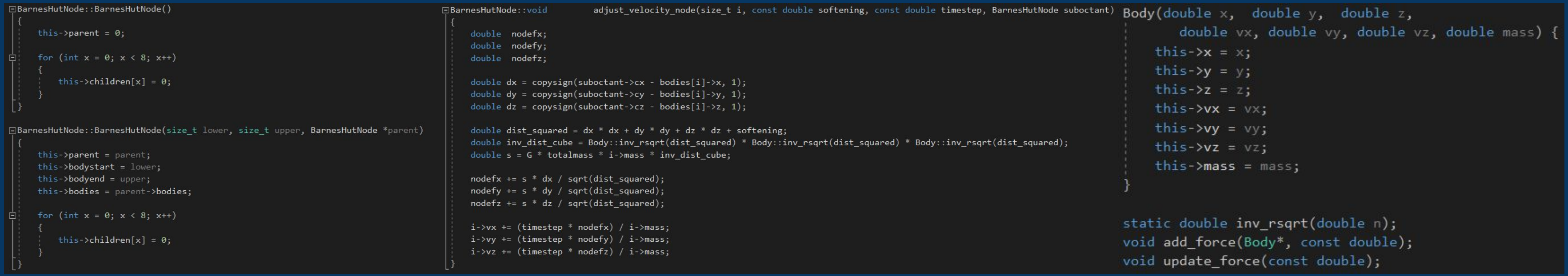
The mass of the bodies in an octant are summed and the center of gravity is found

This allow the calculations to be made on the octant as a whole instead of each body individually



The Program

A text file inputs the position x, y, and z, velocity and mass for a individual body. A spacebody class creates an object from this data. Each body is then passed to an 8-branch tree called and Octree. The Barnes Hut Algorithm then divides these bodies into octants. The gravitational forces and new velocities are calculated from the octants. The new calculated data is then printed to a text file.



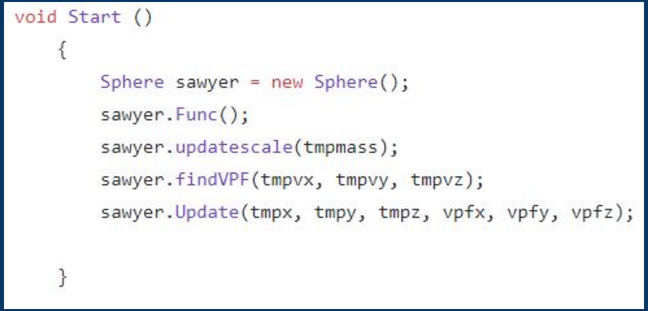
```cpp
BarnesHutNode::BarnesHutNode()
{
    this->parent = 0;

    for (int x = 0; x < 8; x++)
    {
        this->children[x] = 0;
    }
}

BarnesHutNode::BarnesHutNode(size_t lower, size_t upper, BarnesHutNode *parent)
{
    this->parent = parent;
    this->bodystart = lower;
    this->bodyend = upper;
    this->bodies = parent->bodies;

    for (int x = 0; x < 8; x++)
    {
        this->children[x] = 0;
    }
}
```

```cpp
BarnesHutNode::void        adjust_velocity_node(size_t i, const double softening, const double timestep, BarnesHutNode suboctant)
{
    double  nodefx;
    double  nodefy;
    double  nodefz;

    double dx = copysign(suboctant->cx - bodies[i]->x, 1);
    double dy = copysign(suboctant->cy - bodies[i]->y, 1);
    double dz = copysign(suboctant->cz - bodies[i]->z, 1);

    double dist_squared = dx * dx + dy * dy + dz * dz + softening;
    double inv_dist_cube = Body::inv_rsqrt(dist_squared) * Body::inv_rsqrt(dist_squared) * Body::inv_rsqrt(dist_squared);
    double s = G * totalmass * i->mass * inv_dist_cube;

    nodefx += s * dx / sqrt(dist_squared);
    nodefy += s * dy / sqrt(dist_squared);
    nodefz += s * dz / sqrt(dist_squared);

    i->vx += (timestep * nodefx) / i->mass;
    i->vy += (timestep * nodefy) / i->mass;
    i->vz += (timestep * nodefz) / i->mass;
}
```

```cpp
Body(double x,  double y,  double z,
     double vx, double vy, double vz, double mass) {
    this->x = x;
    this->y = y;
    this->z = z;
    this->vx = vx;
    this->vy = vy;
    this->vz = vz;
    this->mass = mass;
}

static double inv_rsqrt(double n);
void add_force(Body*, const double);
void update_force(const double);
```

Unity

The raw data from the text file is then sent to a Unity game engine to create a simulation of the bodies interactions on a gravitational level.

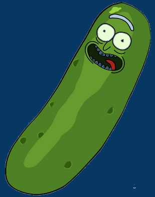


```
void Start ()
    {
        Sphere sawyer = new Sphere();
        sawyer.Func();
        sawyer.updatescale(tmpmass);
        sawyer.findVPF(tmpvx, tmpvy, tmpvz);
        sawyer.Update(tmpx, tmpy, tmpz, vpfx, vpfy, vpfz);

    }
```

Roles

Sawyer - Team Leader/Head Programmer/Expert

Robert -Physics/Math/Barnes-Hut implementation



Cristian - Input/Output/C# Unity code/Moral Support

Odysseas - Octree/Snack/Distractions