Software and Robotic Integration

Hard Constraints Part 1

Rachel Sparks, Ph.D.
Rachel.sparks@kcl.ac.uk
Lecturer in Surgical & Interventional Engineering
School of Biomedical Engineering & Imaging Sciences

Overview of Path Planning

Moving objects through space, including robots and other medical tools, should take into account:

- Physical constraints
- Patient safety
- Medical staff safety

Path planning is how to ensure the robot/tool moves without violating the laws of physics (or poking out an eye)

Overview of Path Planning

Informally, path planning is often described by hard constraints

- Cannot go through <bone, surgical tables, surgeon>
- Avoid <vessels, lung>
- Target <tumour>; Insert into <kidney>

These can also be inequalities

- Drill bits slip at angles above <degree> to <bone>
- Due to breathing motion we avoid <vessels> by up to <distance> to ensure safe targeting

Overview of Path Planning

Path planning also involves soft constraints

- Minimize <length of tool> in <tissue>
- Maximize <distance> to <vessel>
- Reduce <time> to <position robot>

These will be discussed in more detail next week

Path Planning Formalization

Path planning involves

- Potential paths (states) a robot can take $S(t)$
  - A state $s(t) \in S(t)$ is a function that defines the location in the world the object occupies
  - t is time, parameterizing changes in robot state over time

States can be either discrete variables, e.g. the robot an move 1mm in the x, y, or z direction, or a continuous function based on some input parameters, e.g. my robot can move to any point x within a cube defined by [x_min, x_max]. If it is continuous we can assume there are a set of parameters θ that define the position of the robot making S(t) -> S(t, θ)

Path Planning Formalization

Path planning involves

- Potential paths (states) a robot can take $S(t)$
- An image scene B that contains a set of objects $l \in L$
    - Many image segmentation algorithms (see Week 1 Lecture)
    - The scene is represented as $f_B(c, l) = [0, 1]$
    - Each pixel c has a binary value for all $l \in L$

Strictly speaking the objects can be time varying too! This makes our life a lot more complicated, so we are going to ignoring it at the moment

Path Planning Formalization

Path planning involves

- Potential paths (states) a robot can take $S(t)$
- An image scene B that contains a set of objects $l \in L$
- Hard constraints that must be met:
    - Cannot go through <bone>

      $S_{good}(t) \subseteq S(t)$ s.t. $Intersect(s(t), f_B(c, l_{bone}) = 1) = 0 : \forall s(t) \in S_{good}(t)$
    - Target <tumour>

      $S_{good}(t) \subseteq S(t)$ s.t. $Intersect(s(t), f_B(c, l_{tumour}) = 1) \neq 0 : \forall s(t) \in S_{good}(t)$
    - Path must have an angle with the skull below 35°

      $S_{good}(t) \subseteq S(t)$ s.t. $f_a(s(t), f(c, l_{skull}) = 1) < 35° : \forall s(t) \in S_{good}(t)$

- ~~Soft constraints that should be optimized for:~~

Strictly speaking the objects can be time varying too! This makes our life a lot more complicated, so we are going to ignoring it at the moment

Path Planning Formalization

Path planning involves

- Potential paths (states) a robot can take $S(t)$
- An image scene B that contains a set of objects $l \in L$
- Hard constraints that must be met:

We want the following algorithm

```
For s(t) ∊ S(t)
   For c ∊ B
          if (Criteria 1 & Criteria 2 & Criteria 3)
             S_good(t) ← s(t)
```

What is the O?

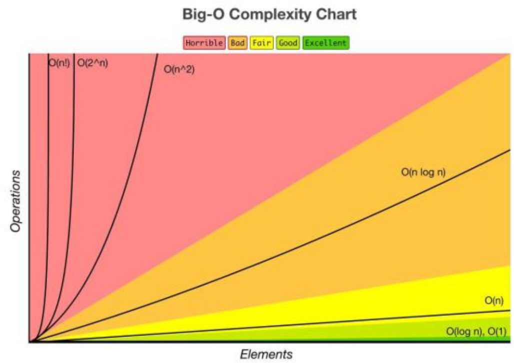
What is the Big O notation for t his most simply of formulas? O(N_s N_c N_critieria) in general the number of elements in b is going to be the rate limiting step; although in some complex robotic configurations it may be the number of states

Big “O” Notation

Big “O” gives a sense of the theoretical worst case scenario

- Can help identify software performance early independent of implementation details
- Useful to select best algorithm before implementation

Some things wont change your Big “O” but will help your algorithm go faster on average



Path Planning Formalization

We want to design the following algorithm

```
For s(t) ∊ S(t)
   For c ∊ B
         if (Criteria 1 & Criteria 2 & Criteria 3)
            S_good(t) ← s(t)
```

This algorithm is

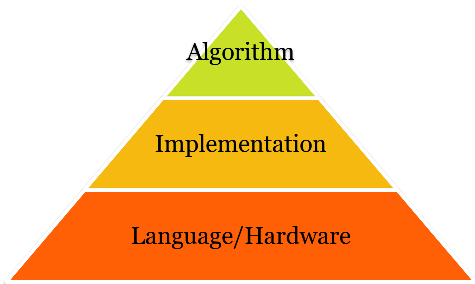
- Incredibly slow and computationally intensive but…
- (As long as Criteria are formulated correctly) gives the complete & correct result
- This makes it great for testing/verification

Everything else today is about what we can do to make this algorithm "better"

Better can be – faster, less computationally intense, reducing the search space (i.e. only exploring subsets of the data).

Algorithm Development – Avoid Over Design

1. Is a better architecture even necessary?
   a. Only to prove a theoretical concept?
   b. Only ever going to run on small N?
   c. Another step in the workflow that is much slower? Re-design the “slowest” steps first
2. Re-design the right thing – try to design “top down”



Path Planning Algorithm Design

We want to design the following algorithm

```
For s(t) ∈ S(t)
   For c ∈ B
          if (Criteria 1 & Criteria 2 & Criteria 3)
             S_good(t) ← s(t)
```

What can we make better here?

Path Planning Algorithm Design

We want to design the following algorithm

```
For s(t) ∊ S(t)
   For c ∊ B
          if (!Criteria 1)
             return
          else if (!Criteria 2)
             return
          else if (Criteria 3)
             S_good(t) ← s(t)
```

O($N_s N_b 3$) but will on average exit earlier now – once a single hard constraint is violated.

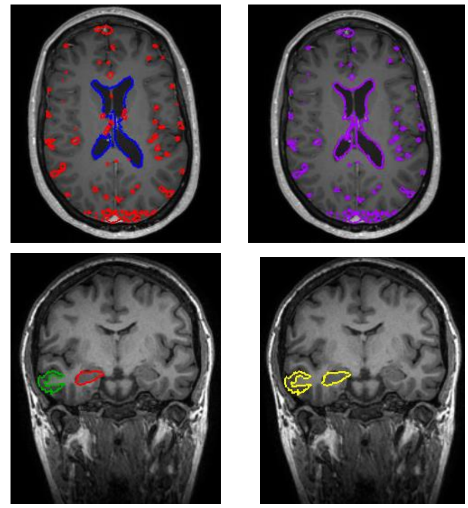
Path Planning Algorithm Design

Consider the details of Criteria and B

- If two criteria are avoid ventricles ($f_B(c, l_{vents}) = 0$) and avoid blood vessels ($f_B(c, l_{vessels}) = 0$)
- Equivalent to avoid all critical structures ($l_{crit} = l_{vents} \cup l_{vessels}$)

Be careful not all criteria can be easily combine

- Place tool through hippocampus ($f_B(c, l_{hippo}) = 1$) and middle temporal gyrus ($f_B(c, l_{mtg}) = 1$)
- NOT equivalent to through ($l_{combo} = l_{hippo} \cup l_{mtg}$)



Path Planning Algorithm Design

We want to design the following algorithm

```
For s(t) ∈ S(t)
      For c ∈ B
         if (!Criteria 13)
            return
         else if (Criteria 2)
            S_good(t) ← s(t)
```

O($N_s N_b 2$) now

- Reduced run time by 1/3 just by combining a few things!
- Best algorithm design

A Note on Image-based Constraints

Collisions/Intersection

- Detecting if you are inside or outside an object is easy
- Treat image as a lookup table and test if label is 0 or 1

Distance

- Detecting how far you are from an object is more complex than collisions
- Need to compute distance from binary mask
    - Simplest to count number of pixels (Chamfer Distance) will be inaccurate up to the width of a pixel
    - More accurate is to use other distance functions (Danielsson , Maurer)

Angle

- Angle of collision with surface is very complex
- Need to estimate local curvature (Anti-Aliasing Binary Image Filter, Active Contour)

A Note on Image-based Constraints

Arrange criteria to reject easiest to compute

- Collisions/Intersection
- Distance
- Angle

Will help increase average speed time no change in O.

Path Planning Implementation

We want to design the following algorithm

```
For s(t) ϵ S(t)          → How robot position is computed
    For c ϵ B            → How image element is defined
        if (!Criteria 13)
            return
        else if (Criteria 2)
            S_good(t) ← s(t)
```

When considering optimization typically cardinality of B is larger than $S(t)$

We have two elements to design – the robot and the images
Hence B is where we should focus our efforts as it is going to give us bigger gains.

Image Iteration

What is $c \in B$ doing

- Element wise iteration over the array of voxels
- Starts at the corner and visits sequentially

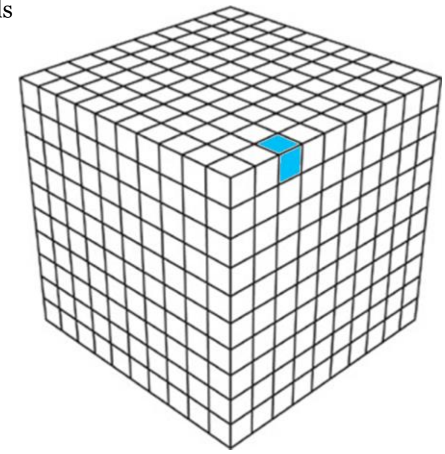


Image Partitioning – Octree Construction

Split B into halves of pixels

- Split is spatial (Left-Right, Anterior-Posterior)
- Now 8 small images
  - Split those into halves
  - Now 64 images
    - Split those into halves
    - .....

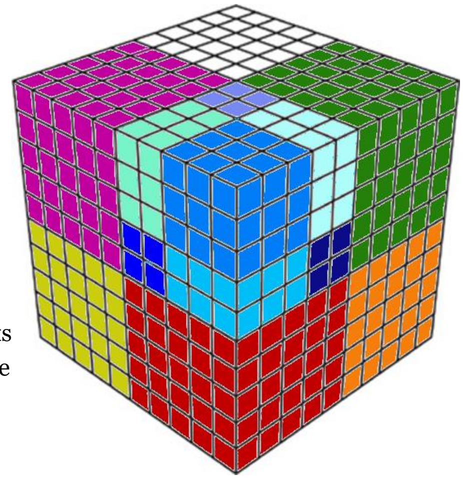


Image divisions grow by 8^n

- By 9^{th} split there are over 100 million elements
- Or enough nodes to fill a 512 x 512 x 512 image

Image Partitioning – Octree Traversal

Start at top most layer

- Do any of the 8 divisions need to be visited?
- Visit only those divisions
  - Do any of their 8 divisions need to visited?
  - Visit only those division?
    - ....

$O(\log(N_B))$ traversal time

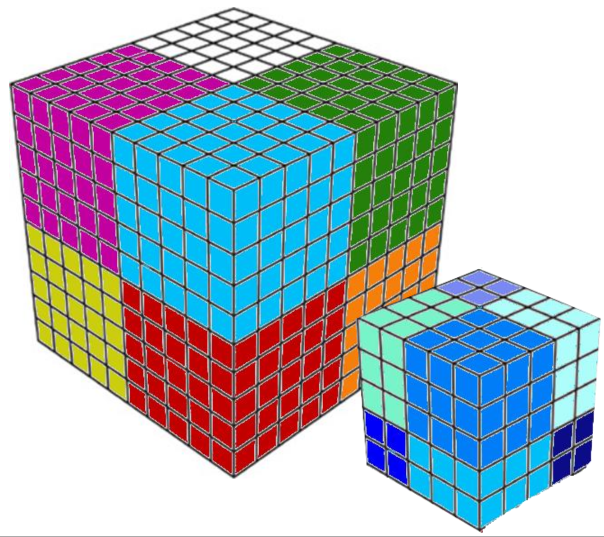


Image Partitioning – Octree Traversal

Enables quick iteration and ignoring of uninteresting portions of our image space

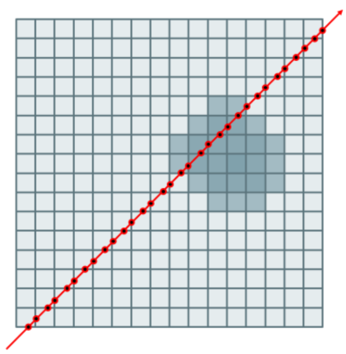
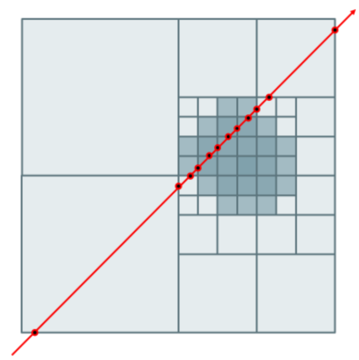


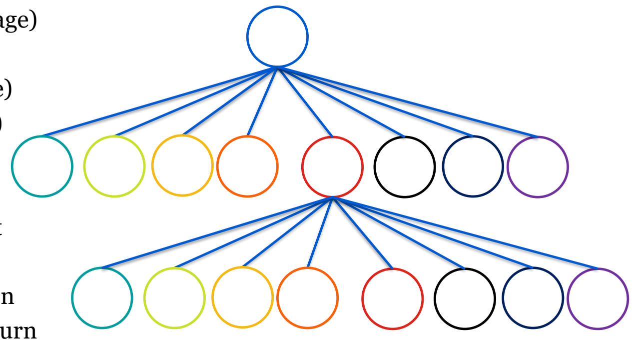
Image Iteration



Octree Iteration

Image Partitioning – Octree Traversal

Easiest to think of an octree as a tree
- "Root" is the top of the tree (i.e. the image)
- Each node may have
  - Children (smaller partitions of image)
  - A parent (a larger partition of image)



Can use recursion for the algorithm
- At each node decide which node to visit next
- When you are at a node with no children ("leaf node") perform some check & return

Image Partitioning – Octree Traversal

Easiest to think of an octree as a tree

- “Root” is the top of the tree (i.e. the image)
- Each node may have
  - Children (smaller partitions of image)
  - A parent (a larger partition of image)

Can use recursion for the algorithm

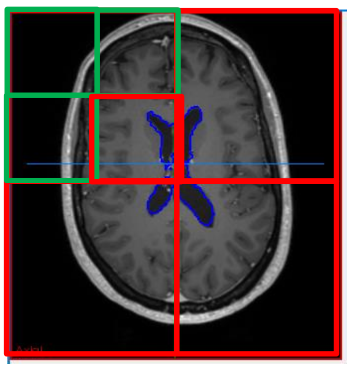
- At each node decide which node to visit next
- When you are at a node with no children (“leaf node”) perform some check & return

```
ClimbThroughTree(Node n, p)
   if !n.HasChildren()
       return collide(n,p)
   else
       Node child = n.GetChild(p)
       ClimbThrougTree(child, p)
```

Image Partitioning – Octree Traversal

Octrees make use of spatial relationships in the image to partition data

- Top down – uses information of the entire image to construct the tree
- Good for detecting collisions when an object is relatively small
- Poor for objects that are dispersed throughout the image



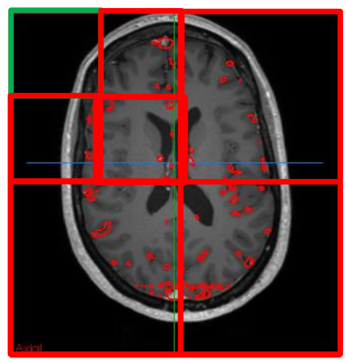
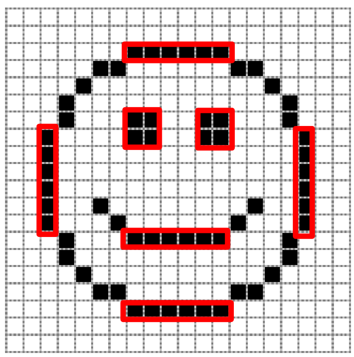


Image Partitioning – Bounding Volume Hierarchy

Another approach is "bottom up"

- Merge neighboring pixels together if they share a label
- Non-cubes are annoying to deal with so only do for "boxes"
- These are our child nodes



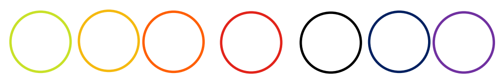
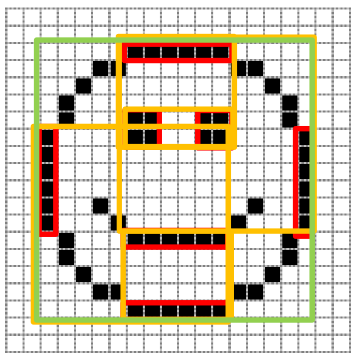


Image Partitioning – Bounding Volume Hierarchy

Another approach is "bottom up"

- Merge closest two children nodes....
- Until no more nodes remain



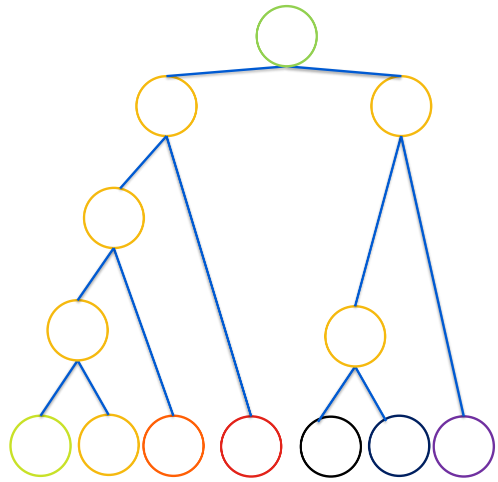


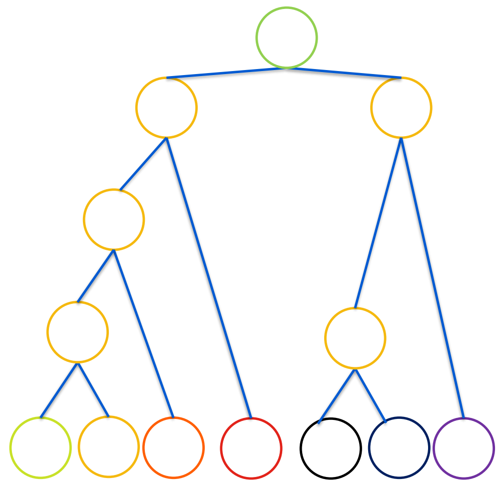
Image Partitioning – Bounding Volume Hierarchy

Another approach is "bottom up"

- Merge closest two children nodes....
- Until no more nodes remain

Just like the octree we can use recursion

- At each node decide which node to visit next
- When you are at a node with no children ("leaf node") perform some check & return



Tree comparison

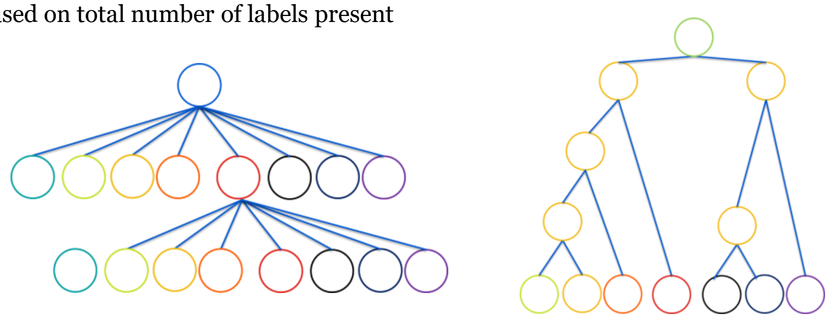
What is the correct tree for your data?

Tree depth

- Octrees are guaranteed balanced – small $\log_8(N_B)$
- BVH can be unbalanced – up to $|l|(|l| - 1)$

Number of nodes

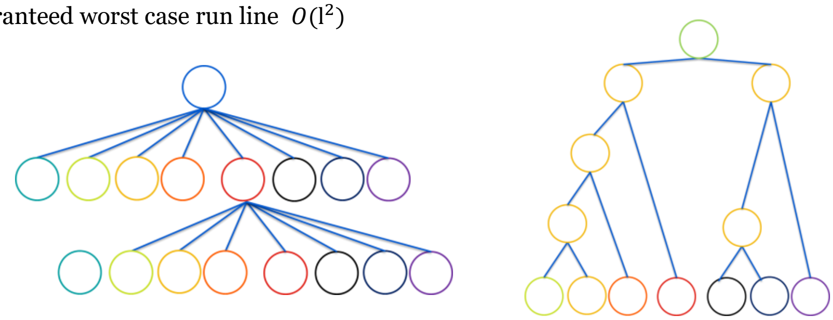
- Octree number of nodes based on total pixel value
- BVH based on total number of labels present



Tree comparison

What is the correct tree for your data?

- Octrees are good for
  - Spatially balanced, sparse data (one solid object)
  - Guaranteed worst case run time $O(\log(N_B))$
- BVHs are good for
  - Spatially unbalanced, sparse data (tendrils)
  - Guaranteed worst case run line $O(\mathrm{l}^2)$



Implementation Notes & Inspiration

ITK Classes

- Image Iterators – Get all pixels
- Conditional Image Iterators – Get all pixels that meet a criteria (say having 1 as the label?)
- Line Iterator – Get all pixels along a line
- Octree – Image octree
- No Bounding Volume ☹

VTK Classes

- Image Iterators – Get all pixels
- Octree – Image octree (plus some extensions)
- Oriented Bounding Box – Only works on mesh data….

Path Planning Implementation

We want to design the following algorithm

```
For s(t) ∊ S(t)
      For c ∊ B
         if (!Criteria 13)
            return
         else if (Criteria 2)
            S_good(t) ← s(t)
```

Iterate over node $\in OT(B)$

Now $O(N_s \log(N_b)2)$

We have two elements to design – the robot and the images
Hence B is where we should focus our efforts as it is going to give us bigger gains.