

Dual Coding Representations and the MapBuilder

15-494 Cognitive Robotics
David S. Touretzky &
Ethan Tira-Thompson

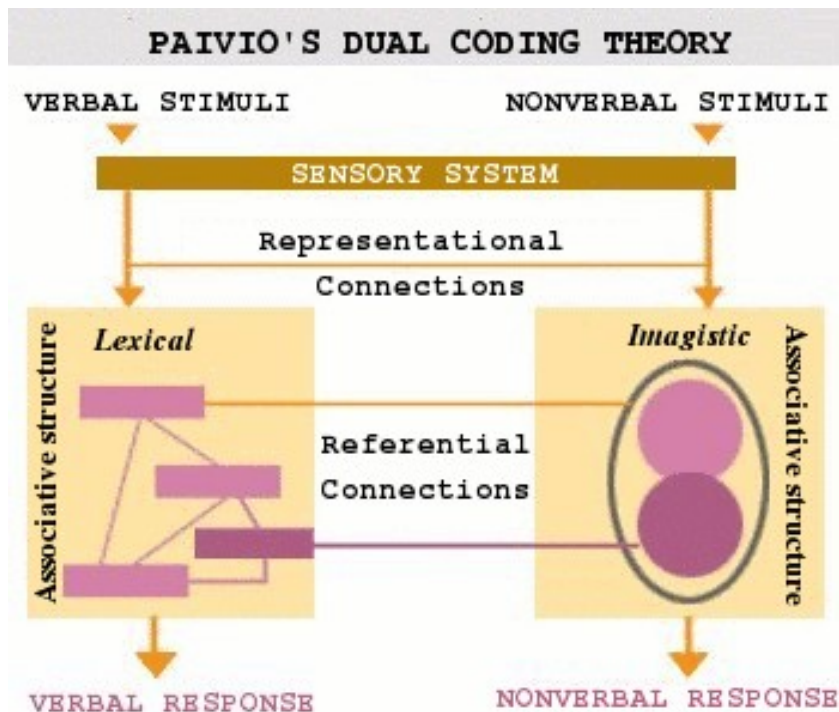
Carnegie Mellon
Spring 2016

Dual-Coding Representation

- Paivio's “dual-coding theory”:

People use both iconic and symbolic mental representations.

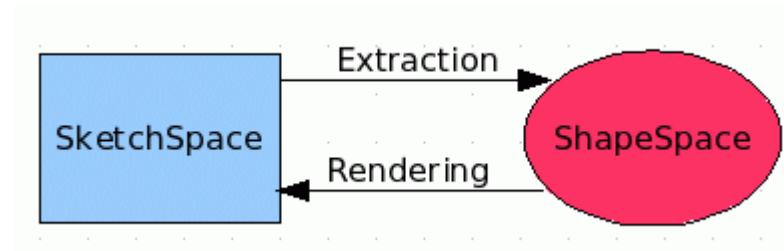
They can convert between them when necessary, but at a cost of increased processing time.



Alan Paivio

Dual-Coding In Tekkotsu

- Tekkotsu implements Paivio's idea:

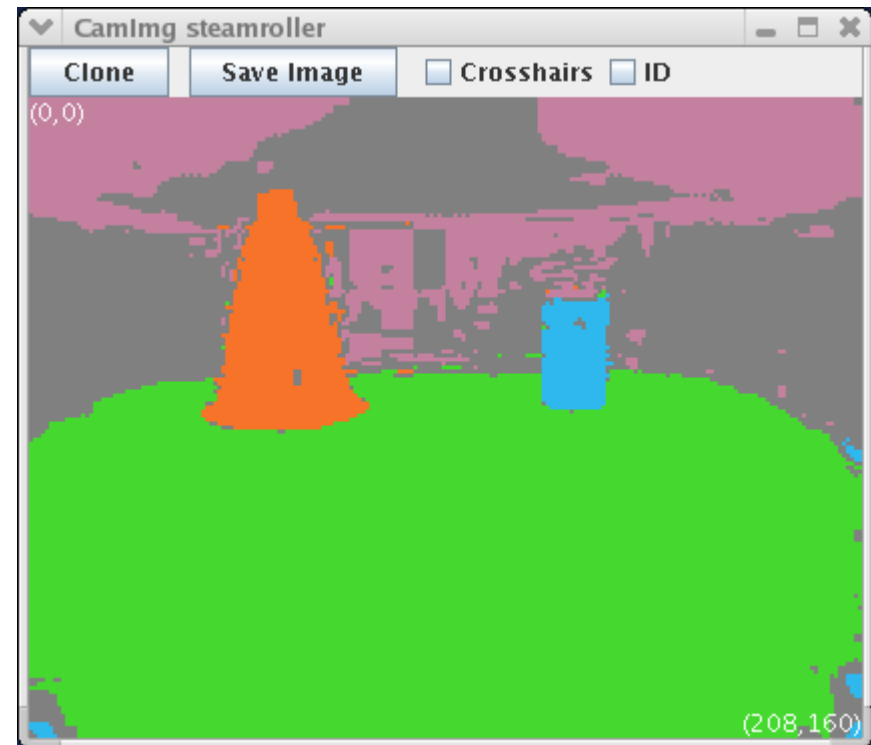
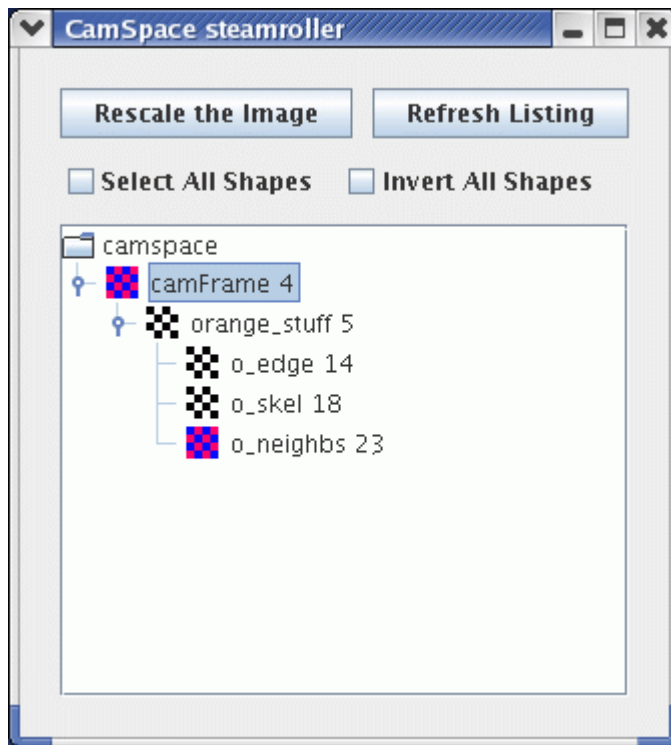


- Sketch space = iconic representation
Shape space = lexical representation
- What would Ullman (inventor of the term “visual routines”) say? Visual routines mostly operate on sketches, but not exclusively.

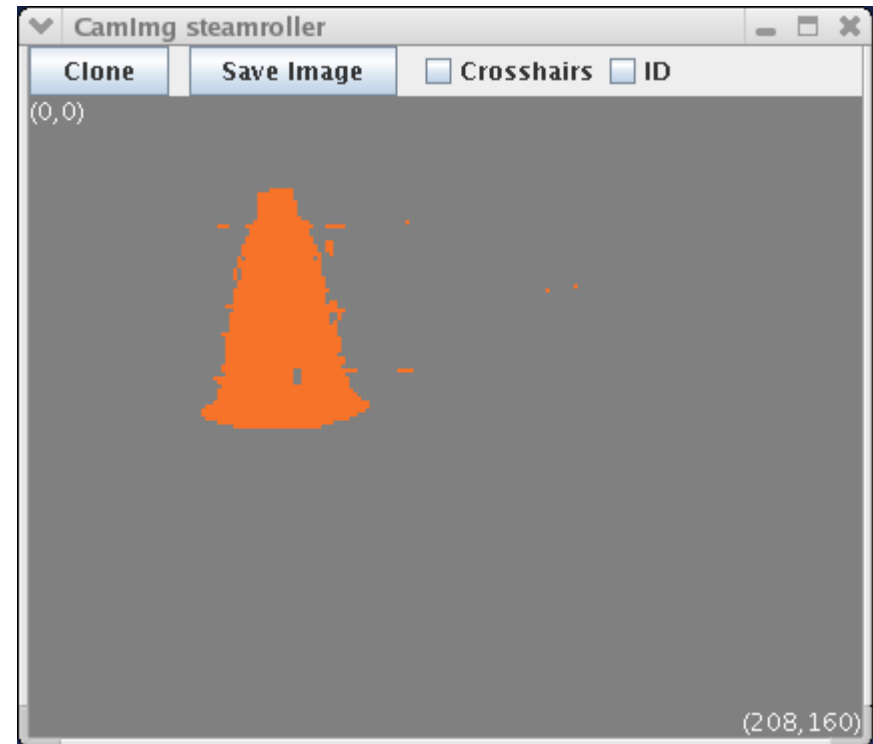
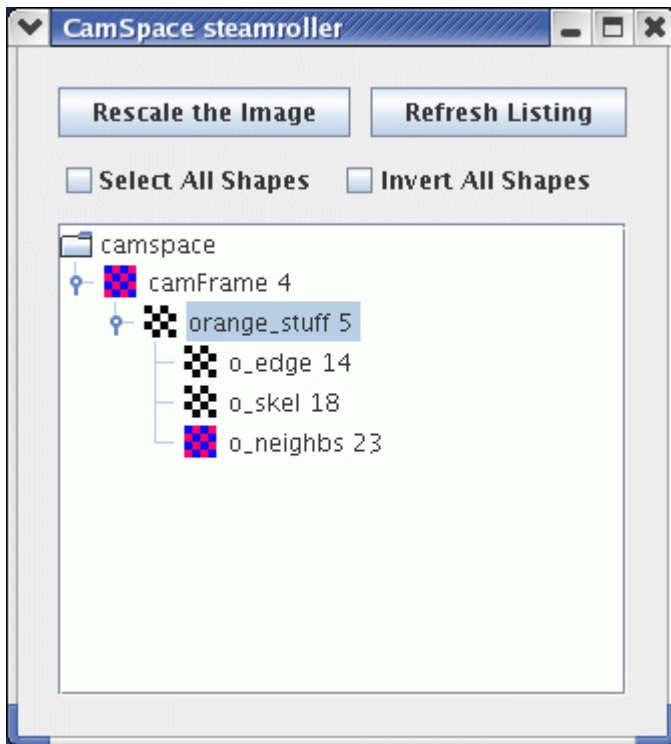
Sketches in Tekkotsu

- A sketch is a 2-D iconic (pixel) representation.
- Templated class:
 - Sketch<uchar> *unsigned char*: can hold a color index
 - Sketch<bool> true if a property holds at image loc.
 - Sketch<uint> *unsigned int*: pixel index; distance; area
 - Sketch<usint> *unsigned short int*
 - Sketch<float> single precision *float*
- Sketches live in a SketchSpace: fixed width and height.
- A built-in sketch space: camSkS.

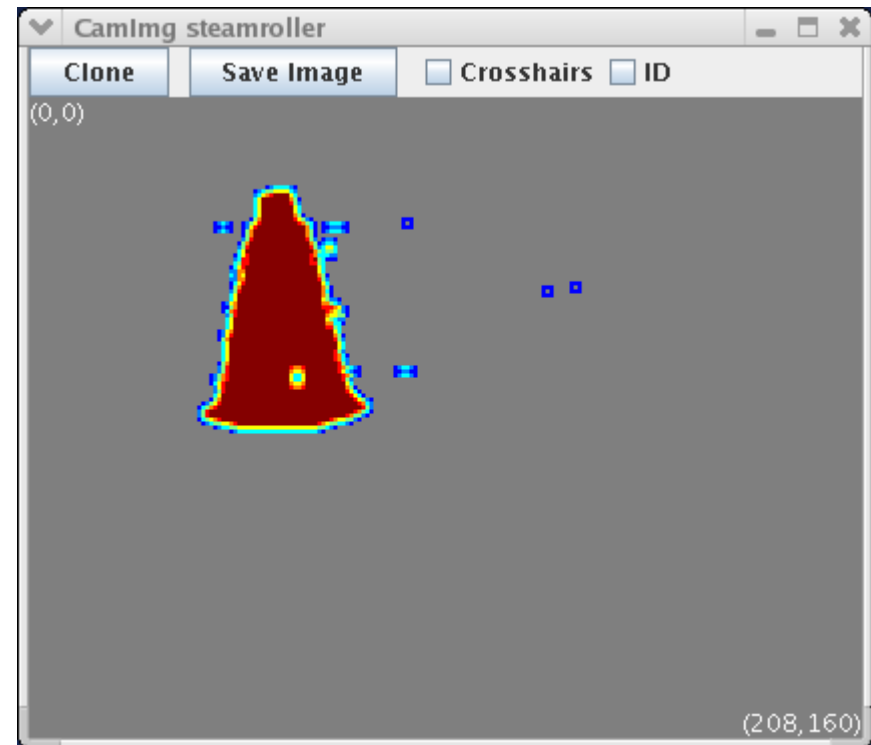
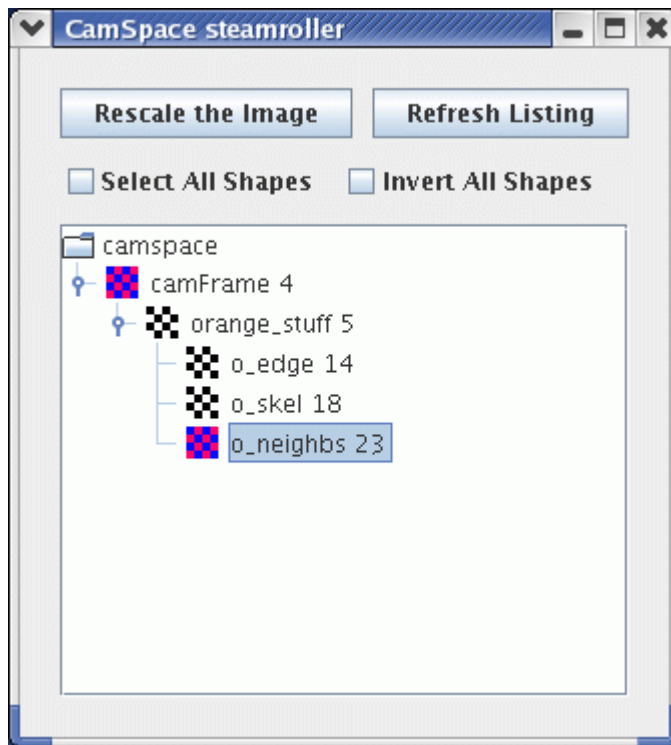
Color-Segmented Image



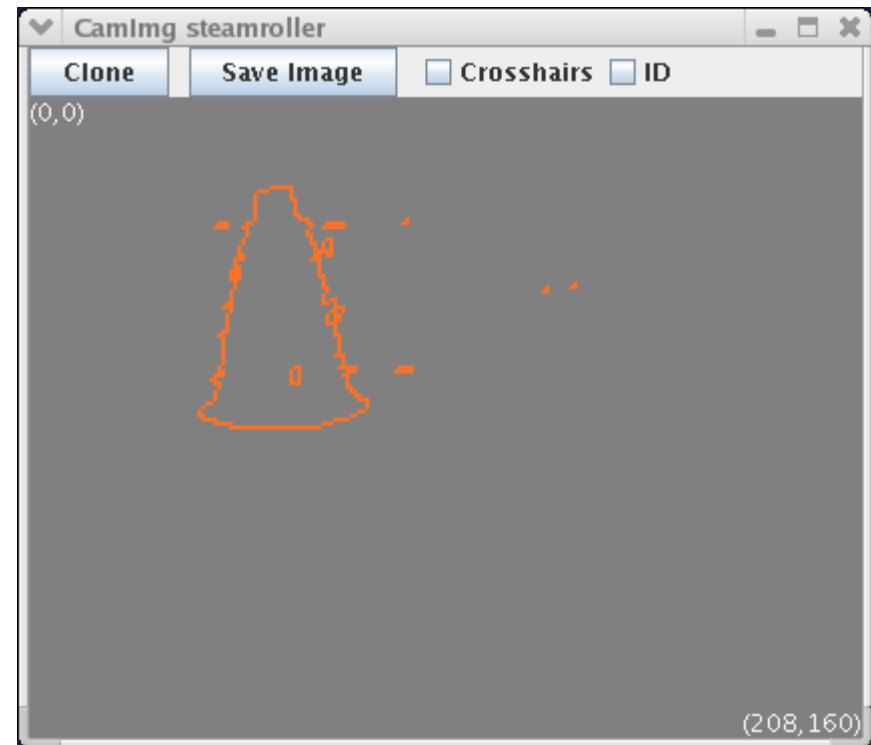
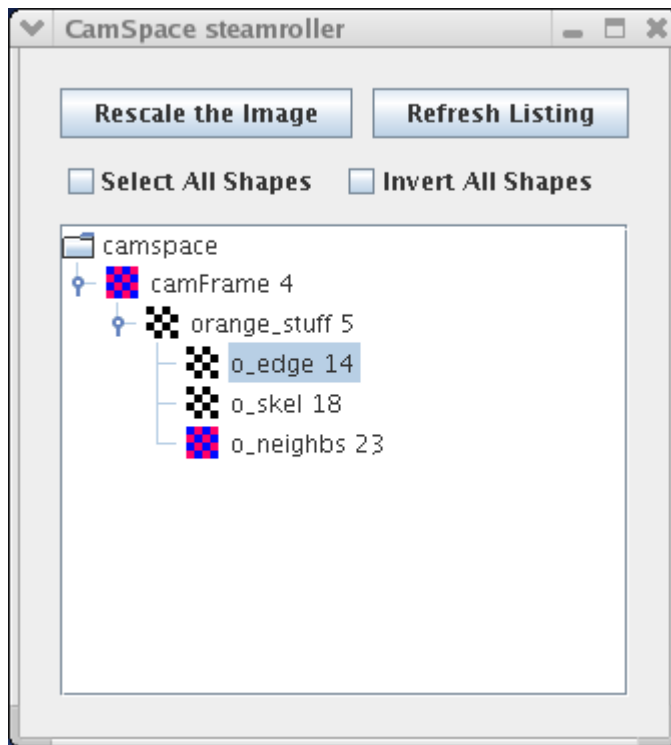
visops::colormask("orange")



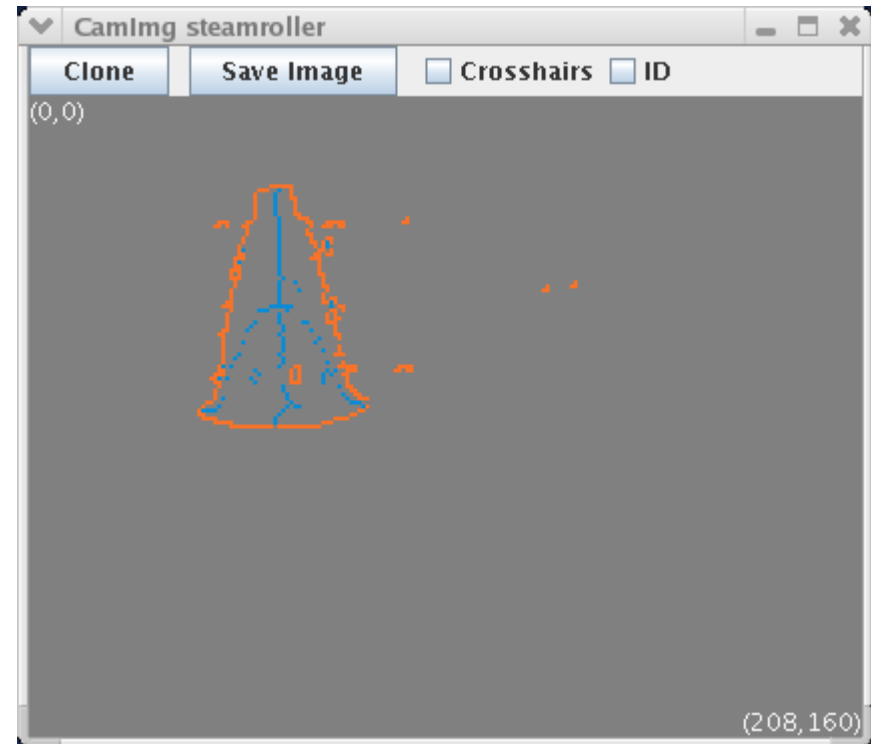
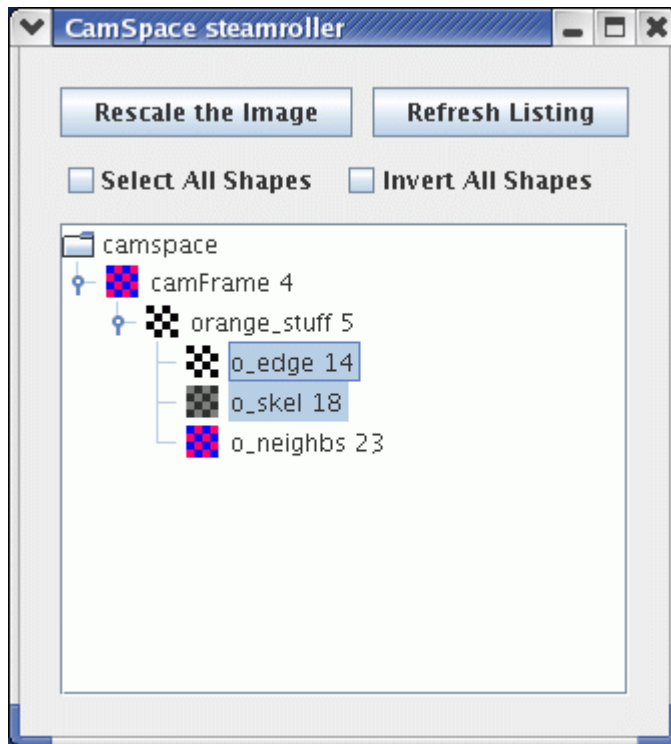
visops::neighborSum(orange_stuff)



visops::edge(orange_stuff)

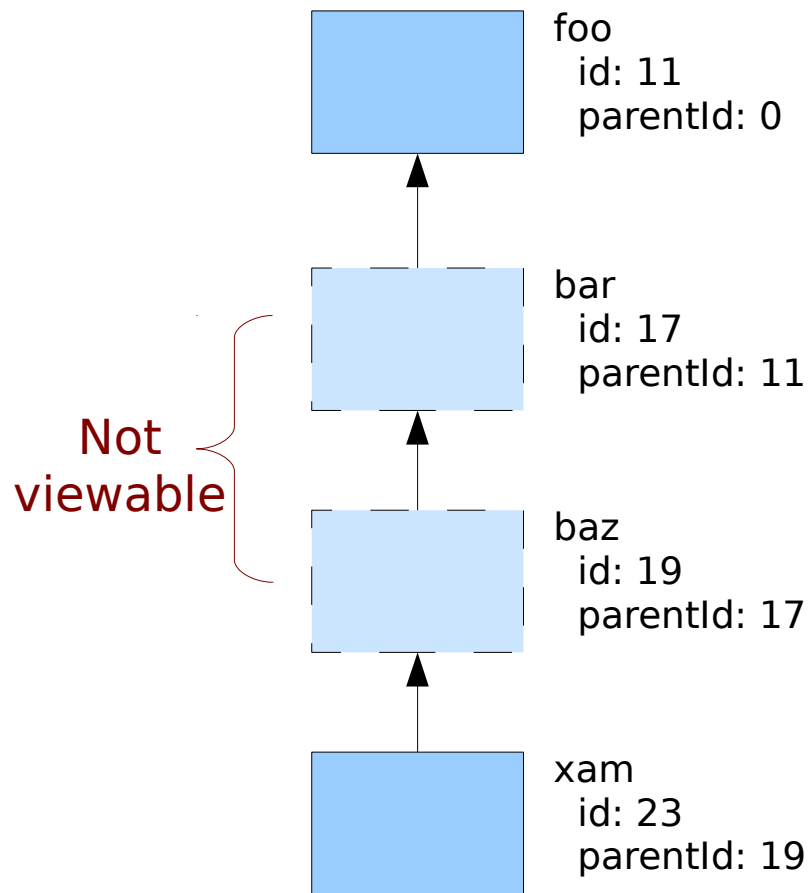


visops::skel(orange_stuff)

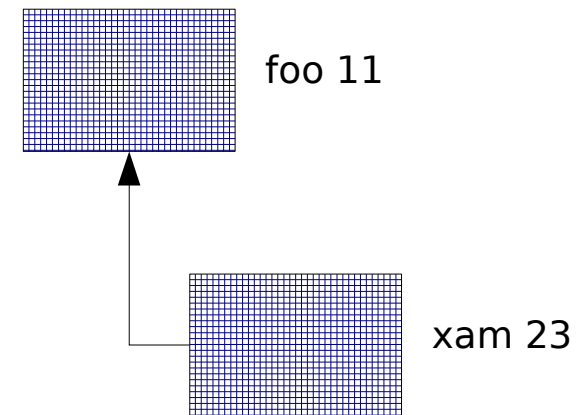


Parents and Viewable IDs

On the Robot



SketchGUI Display

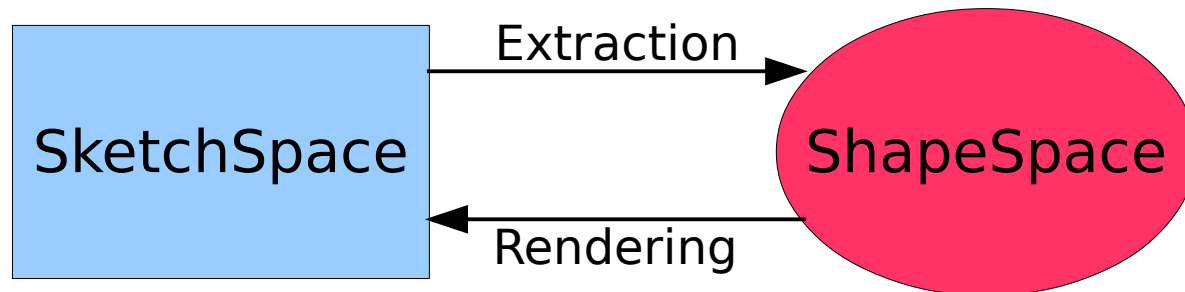


Shapes in Tekkotsu

- Basic types:
 - Line, Polygon
 - Ellipse
 - Blob
- 3D shapes:
 - Sphere, Cylinder, Brick, Pyramid, Domino
- Special purpose:
 - Agent
 - Localization Particle
 - AprilTag, Sift, Marker

Shapes Live in a ShapeSpace

- SketchSpace and ShapeSpace are duals:



- We'll be using camSkS and camShS: the camera sketch and shape spaces.

Some Math For Shapes

- Angles
 - AngTwoPi: angular value from 0 to 2π
 - AngSignPi: angular value from $-\pi$ to π
 - AngPi: angular value from 0 to π
- Vectors and matrices
 - fmat::Column<3>
 - fmat::Transform
- Points (see next slide)

All of these have overloaded arithmetic operators.

Example Shape Constructors

```
LineData(ShapeSpace &space,  
         const Point &p1,  
         const Point &p2)
```

```
EllipseData(ShapeSpace &space,  
            const Point &center,  
            float semimajor,  
            float semiminor,  
            AngPi orientation)
```

Points

- A Point is an object containing:
 - A column vector of coordinates $[x,y,z]$
 - A reference frame type:
 - camcentric
 - egocentric
 - allocentric
 - unknown
- Arithmetic operators: $+ - * /$
 - Checks for reference frame compatibility
- `operator<<` overloaded for convenient printing

Point Arithmetic

```
$nodeclass Ex1 : doStart {  
    Point alpha(50, 75);  
    Point bravo(100, 100, 100, camcentric);  
    Point charlie = alpha + bravo*2;  
    cout << alpha << " + " << bravo << "*2 = "  
        << charlie << endl;  
}
```

Output:

$u: [50, 75, 0] + c: [100, 100, 100]*2 = c: [250, 275, 200]$

Shape<T>

- We don't work directly with LineData and EllipseData objects.
- Instead we work with smart pointers:
 Shape<LineData>
 Shape<EllipseData>
- The smart pointers take care of reference counting and automatic destruction of garbage objects.
- Shape<LineData>() returns an invalid line shape, similar to a NULL pointer.
- To make new shapes we use the NEW_SHAPE macro:

`NEW_SHAPE(name, type, *data)`

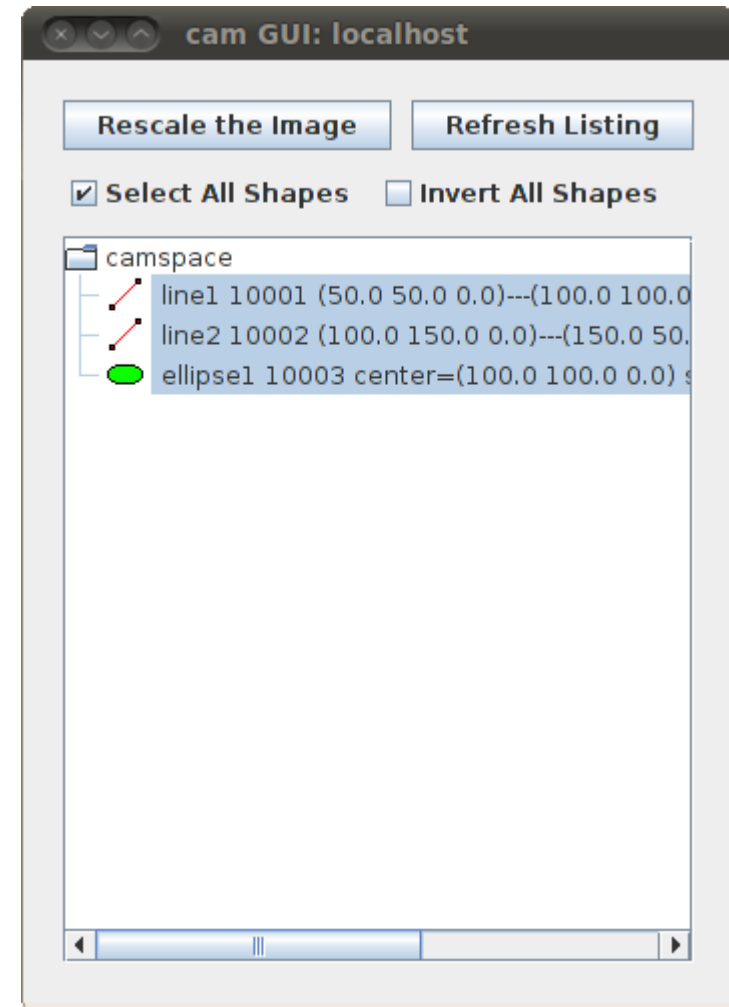
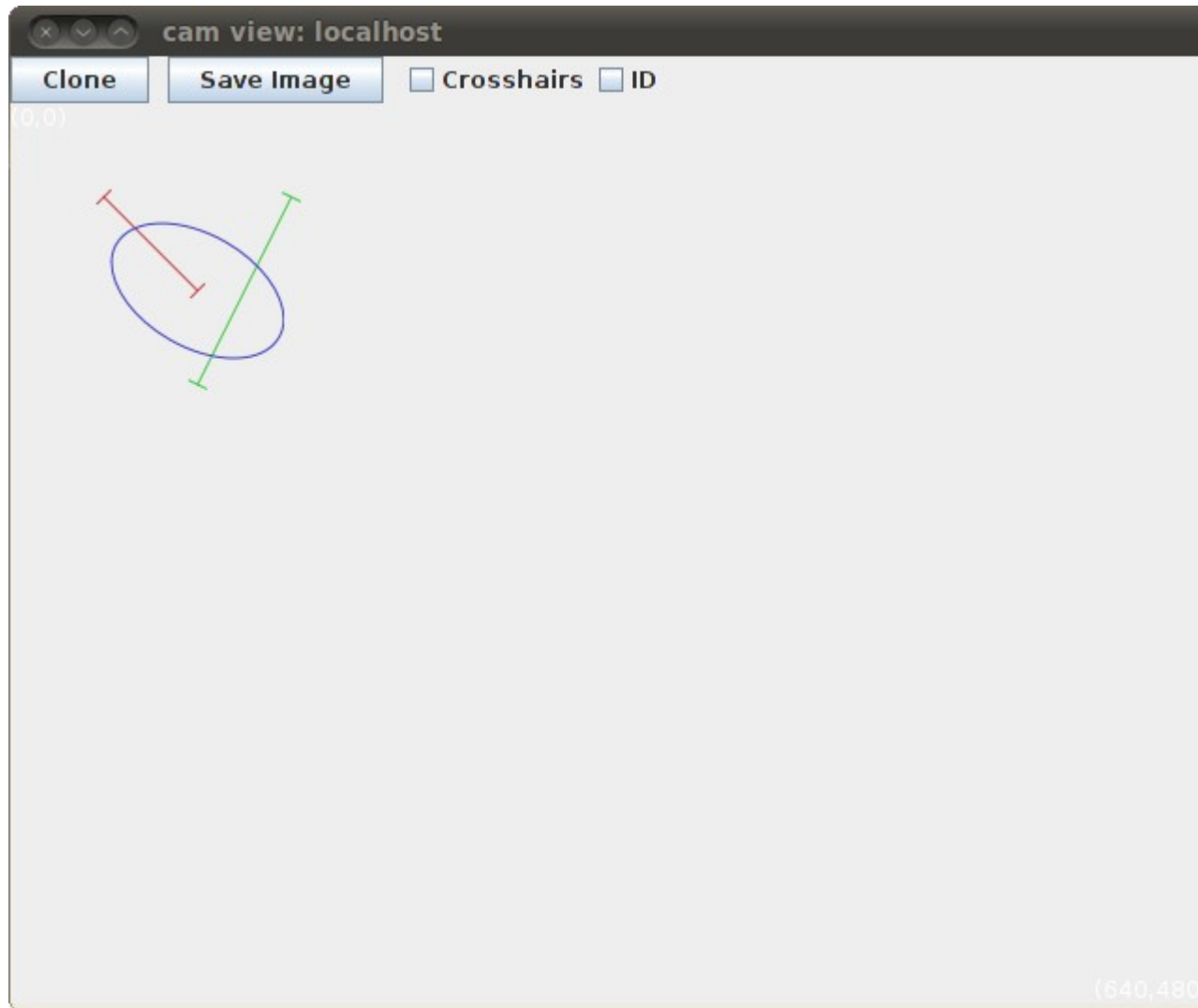
Making New Shapes

```
NEW_SHAPE(line1, LineData,  
    new LineData(camShS, Point(50,50), Point(100,100)));  
line1->setColor("red");
```

```
NEW_SHAPE(line2, LineData,  
    new LineData(camShS, Point(100,150), Point(150,50)));  
line2->setColor("green");
```

```
NEW_SHAPE(ellipse1, EllipseData,  
    new EllipseData(camShS, Point(100,100),  
                    50, 30, M_PI/6));  
ellipse1->setColor("blue");
```

Viewing Our Shapes



NEW_SHAPE Revealed

- NEW_SHAPE is a bit of syntactic sugar:

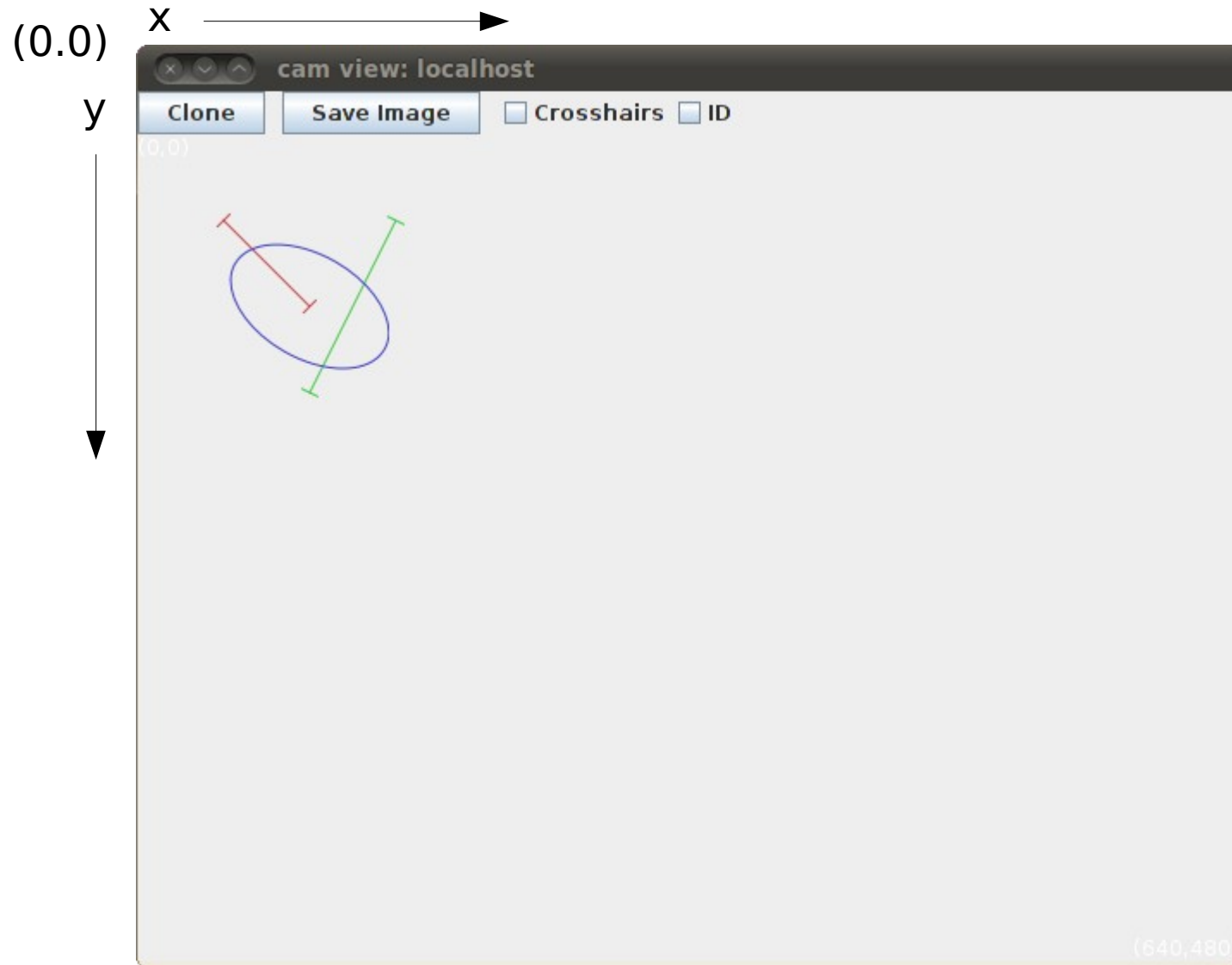
```
NEW_SHAPE(myline, LineData, new LineData(camShS,pt1,pt2))
```

expands into:

```
Shape<LineData> myline(new LineData(camShS,pt1,pt2));  
  
if ( myline.isValid() )  
    myline->V("myline");           // make viewable
```

- Use NEW_SHAPE_N for shapes not to be viewable.

Camera Coordinates



Perceiving Shapes

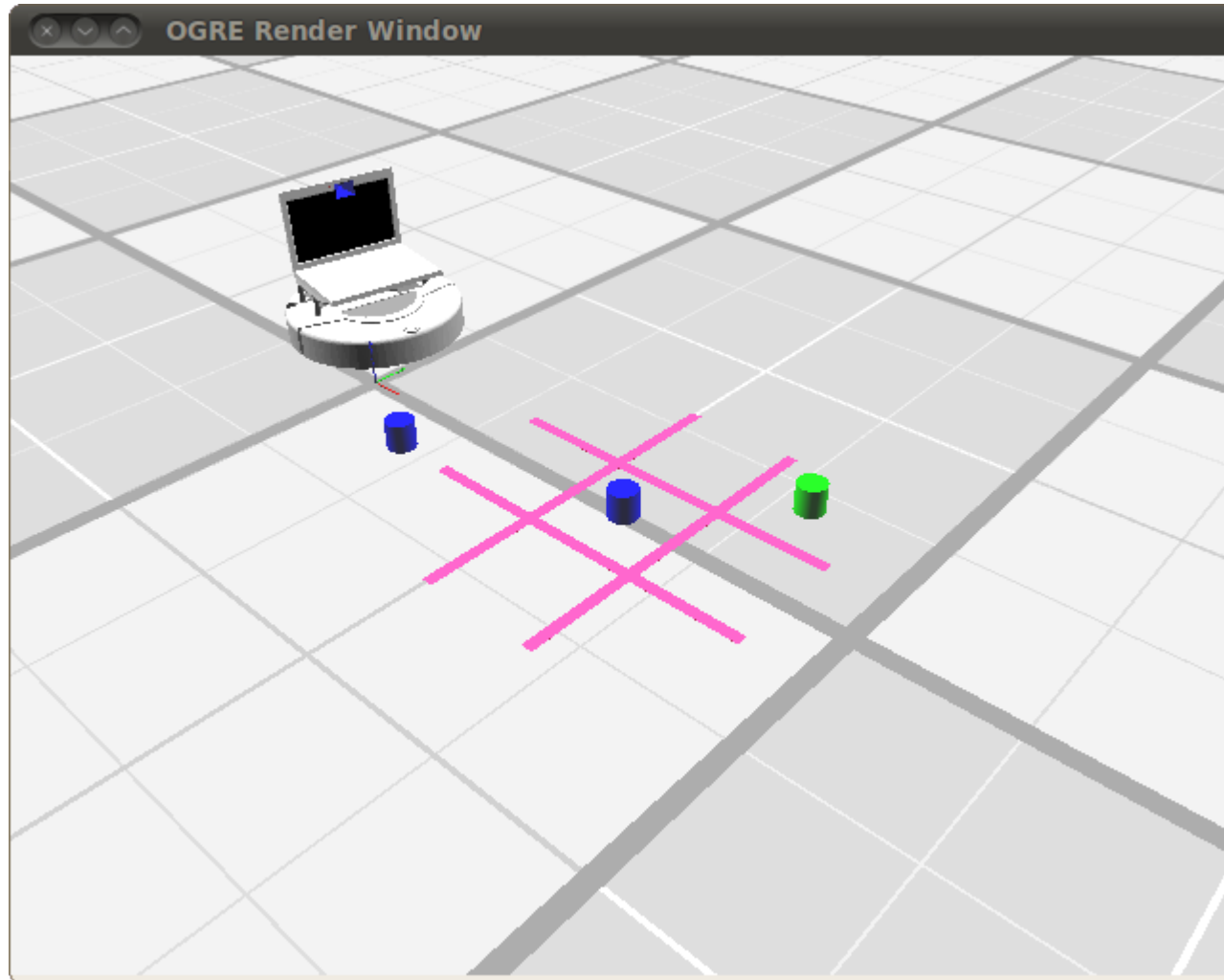
- Rather than making shapes by hand, we want the robot to look at the world and recognize shapes.
- The process works like this:
 - Grab a camera image and encode it as a sketch.
 - Extract various shapes from the sketch and register them in the associated shape space.
- Instead of doing this manually, you can ask the MapBuilder to do it for you.
- A MapBuilderRequest describes what you're looking for.
- Use a MapBuilderNode to construct and submit the request.

Using the MapBuilder

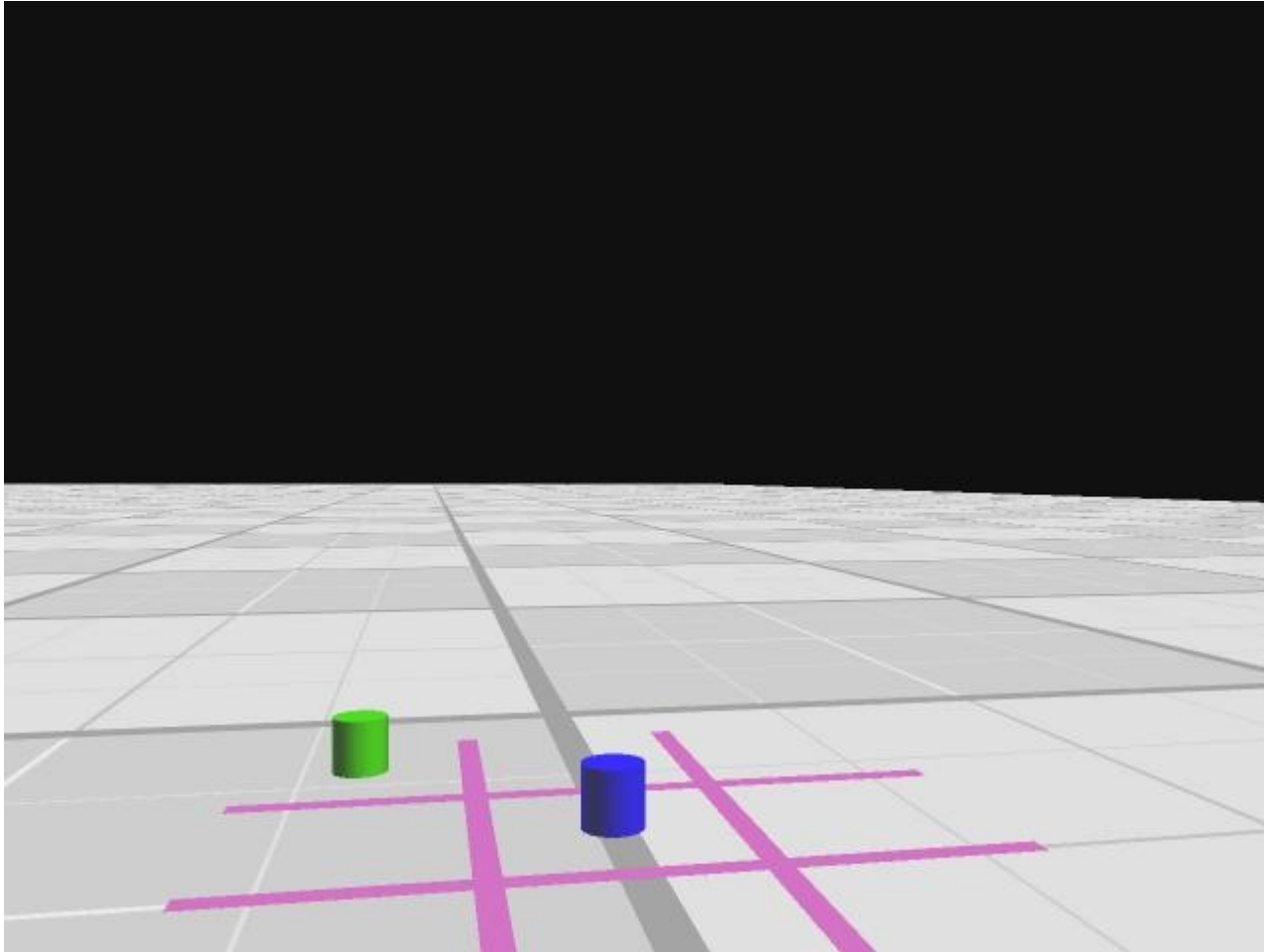
```
$nodeclass Ex2 {  
    $nodeclass FindStuff : MapBuilderNode : doStart {  
        mapreq.addObjectColor(lineDataType, "red");  
        mapreq.addObjectColor(ellipseDataType, "green");  
        mapreq.addObjectColor(ellipseDataType, "blue");  
    }  
    $setupmachine{  
        FindStuff =C=> SpeechNode("done")  
    }  
}
```

Note: **lineDataType** and **ellipseDataType** are defined in Tekkotsu/DualCoding/ShapeTypes.h

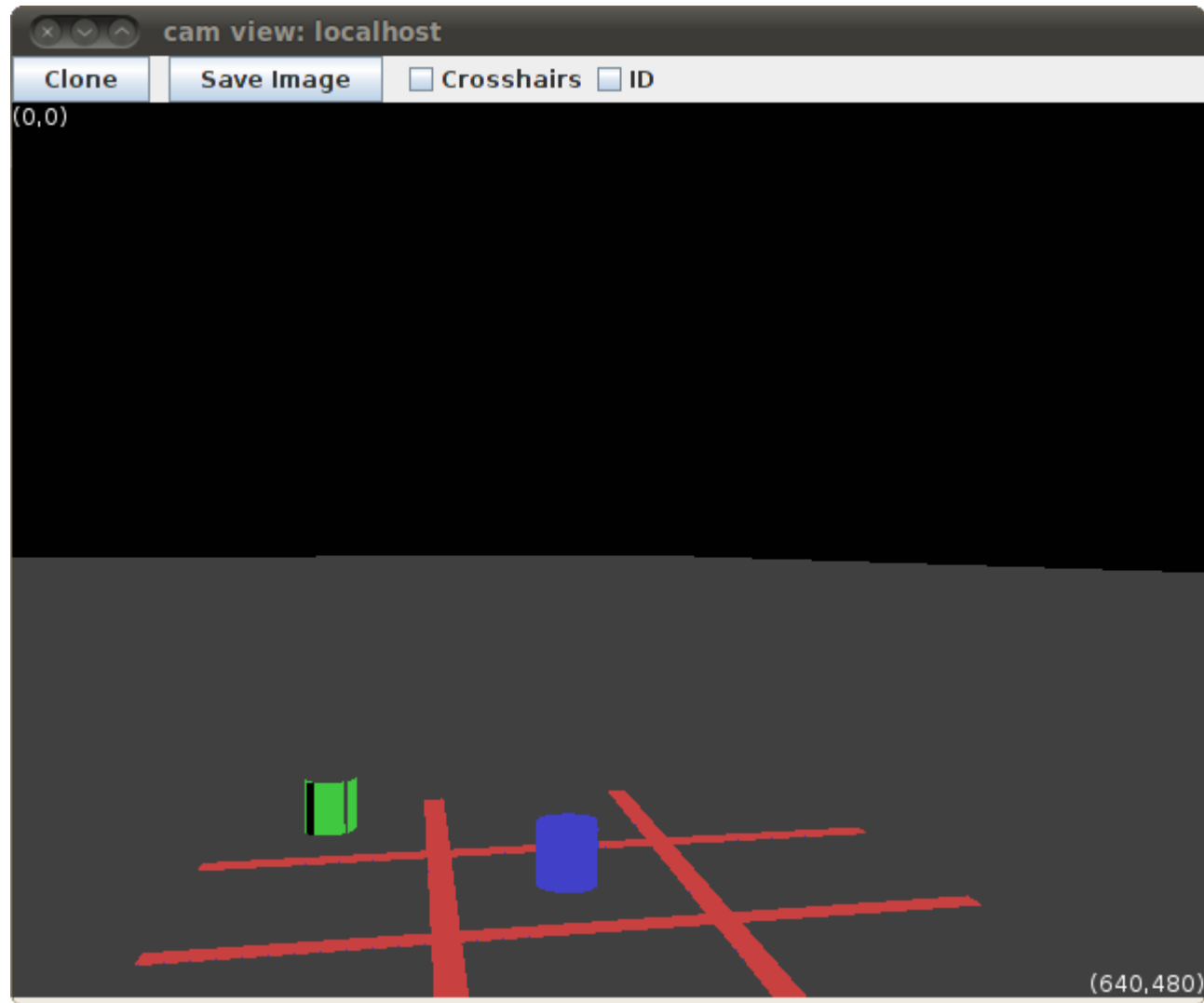
TicTacToe World in Mirage



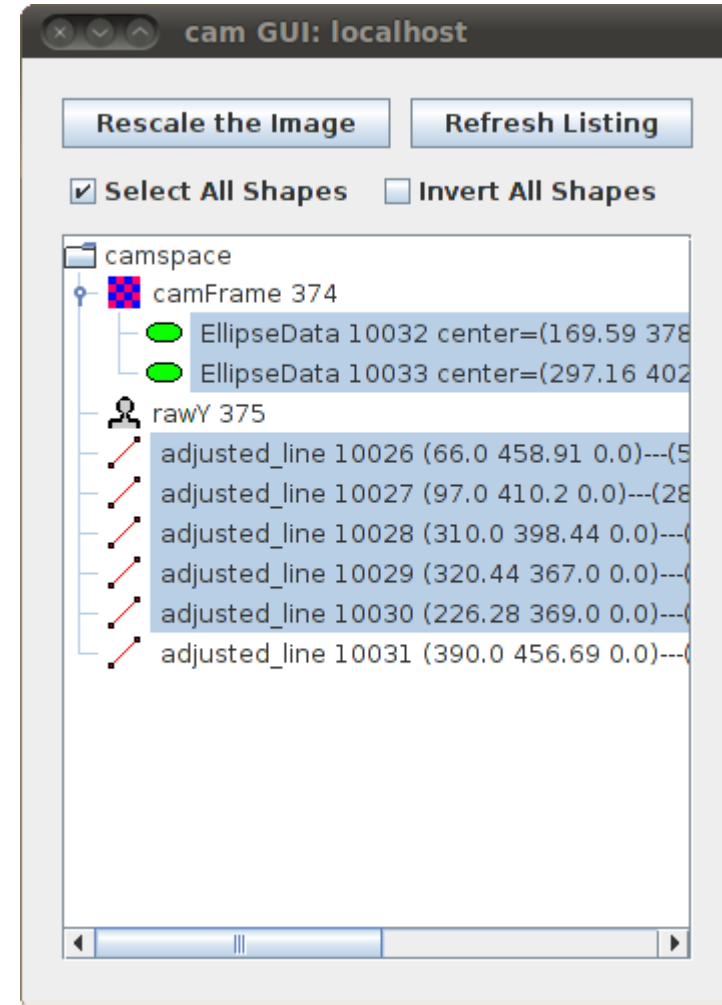
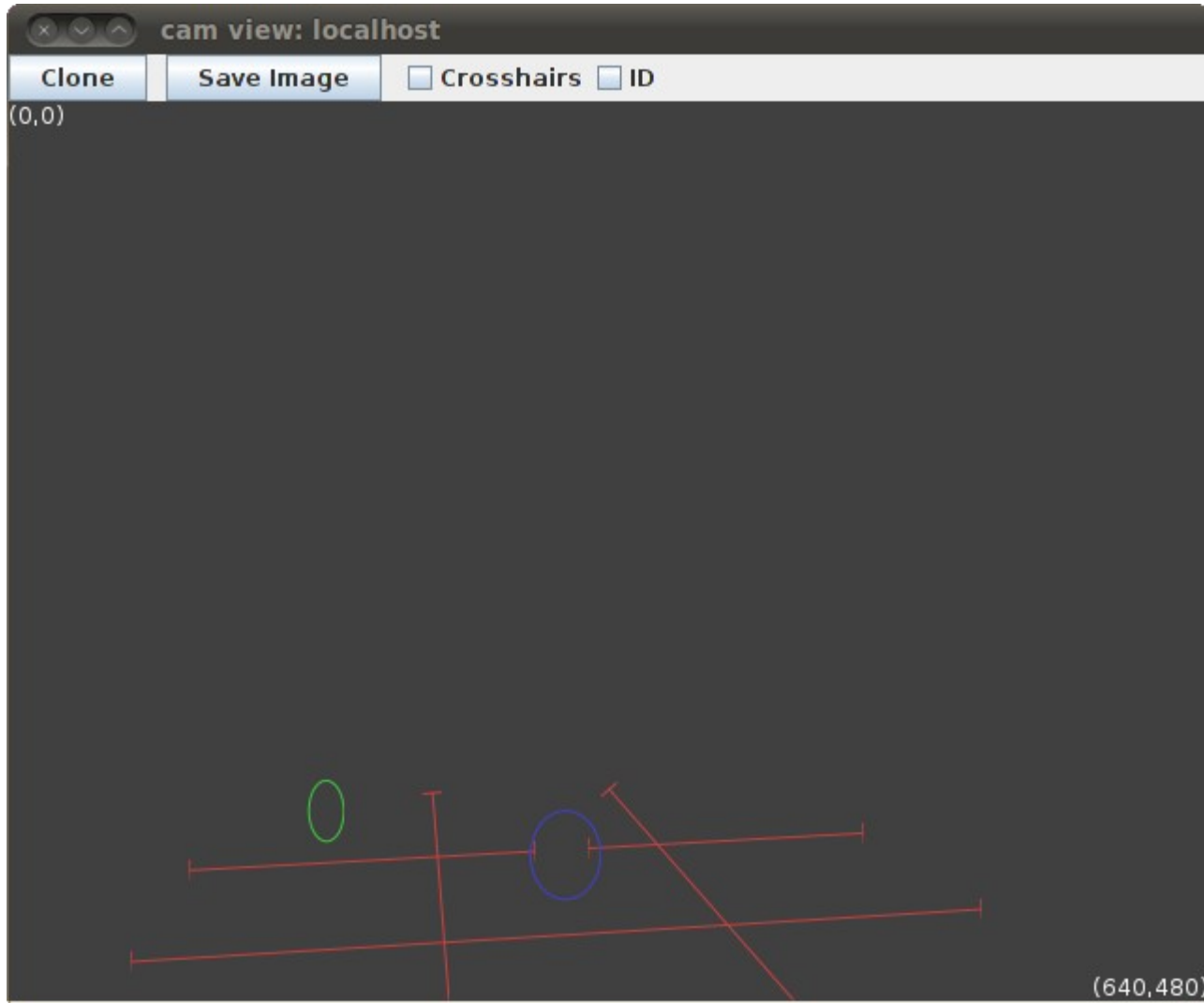
What the Robot Sees



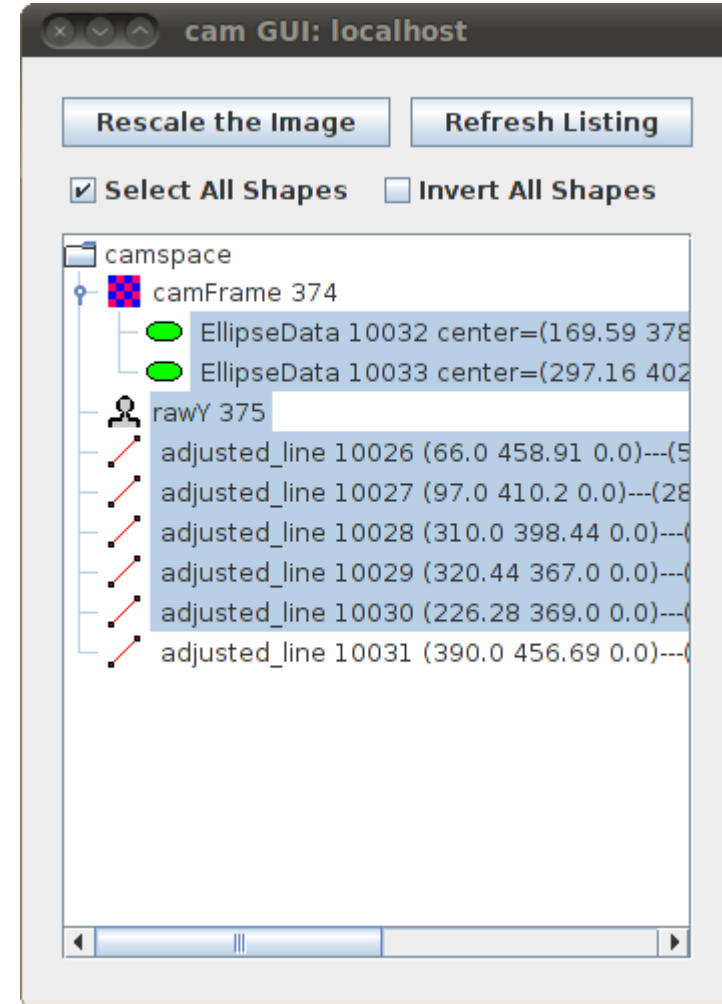
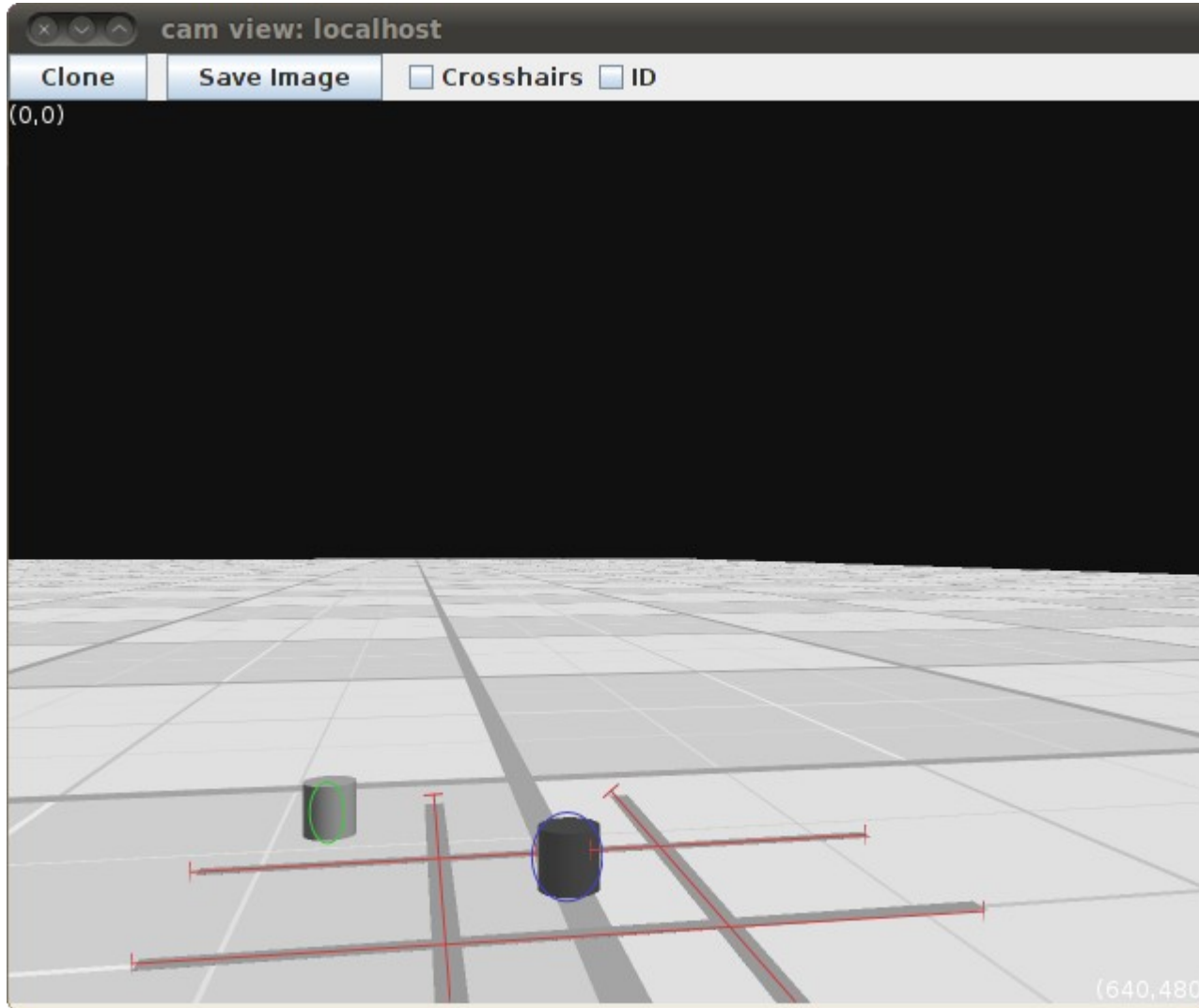
Color Segmented Image



Extracting The Shapes



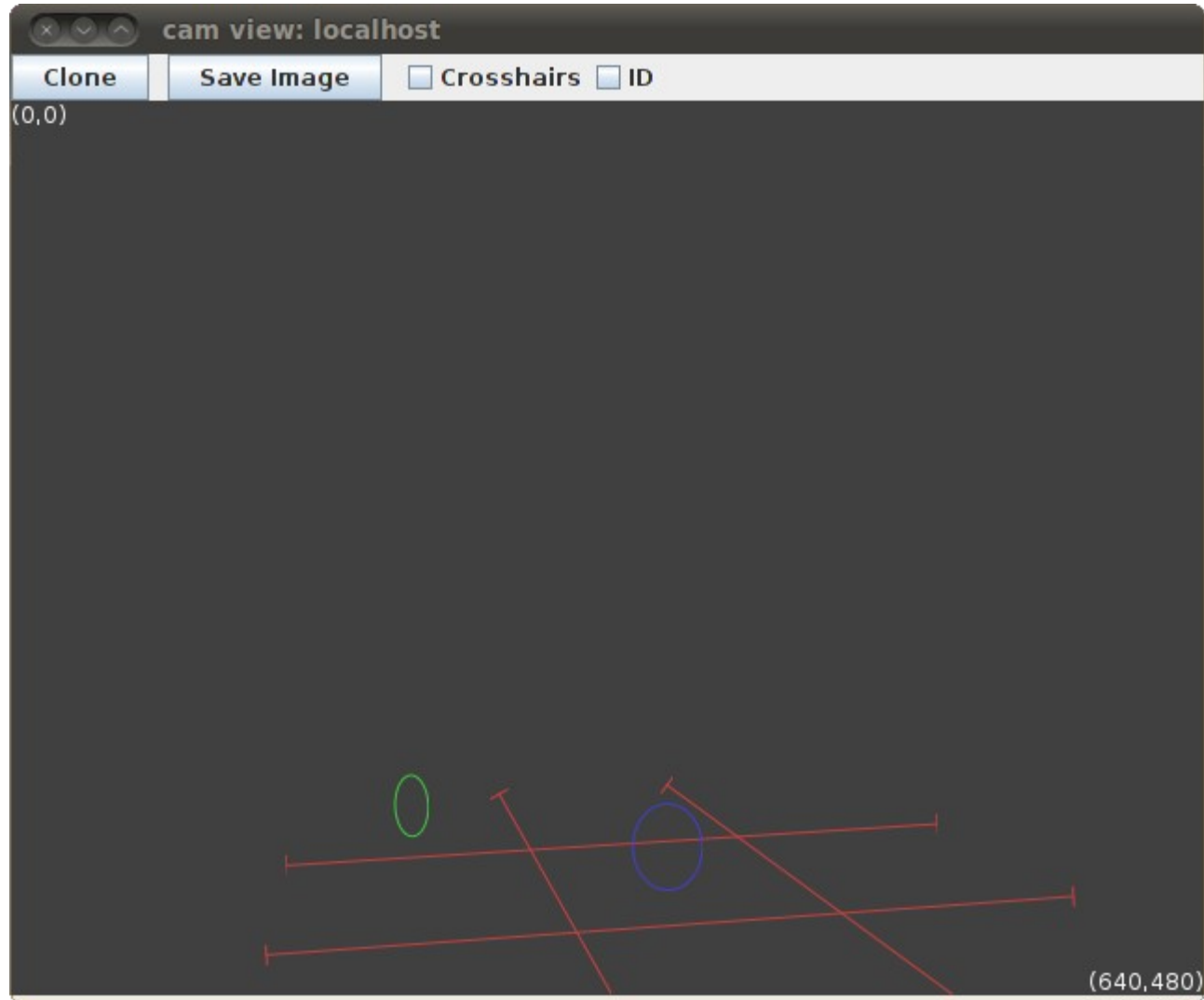
Superimpose RawY Channel



Dealing With Occlusion

```
$nodeclass Ex2 {  
  $nodeclass FindStuff : MapBuilderNode : doStart {  
    mapreq.addObjectColor(lineDataType, "red");  
    mapreq.addObjectColor(lineDataType, "green");  
    mapreq.addObjectColor(lineDataType, "blue");  
    mapreq.addObjectColor(ellipseDataType, "green");  
    mapreq.addObjectColor(ellipseDataType, "blue");  
  }  
  $setupmachine{  
    FindStuff =C=> SpeechNode("done")  
  }  
}
```

Occlusion Resolved

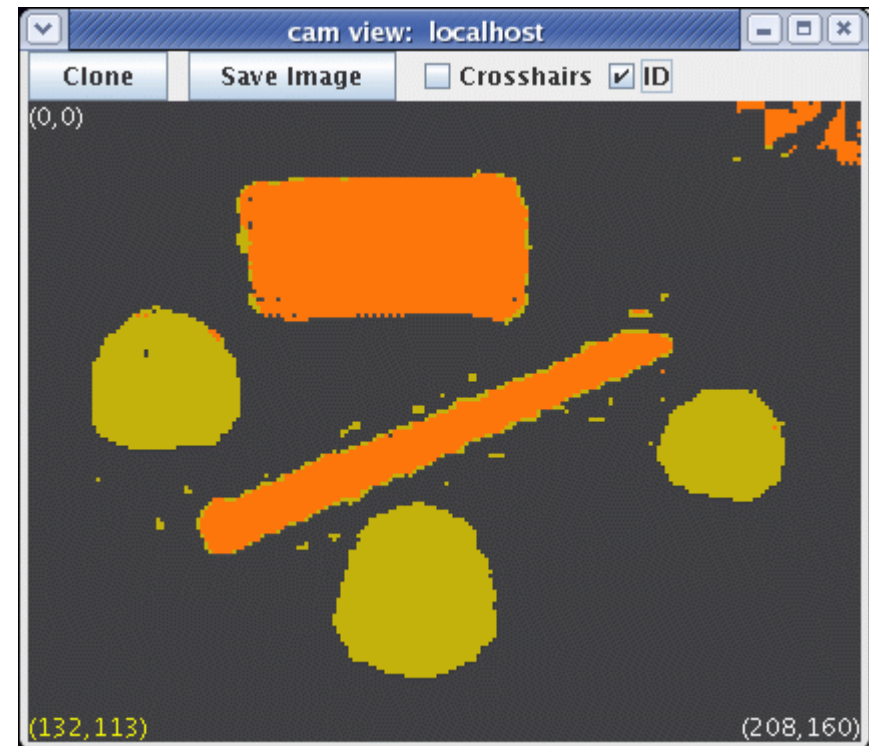
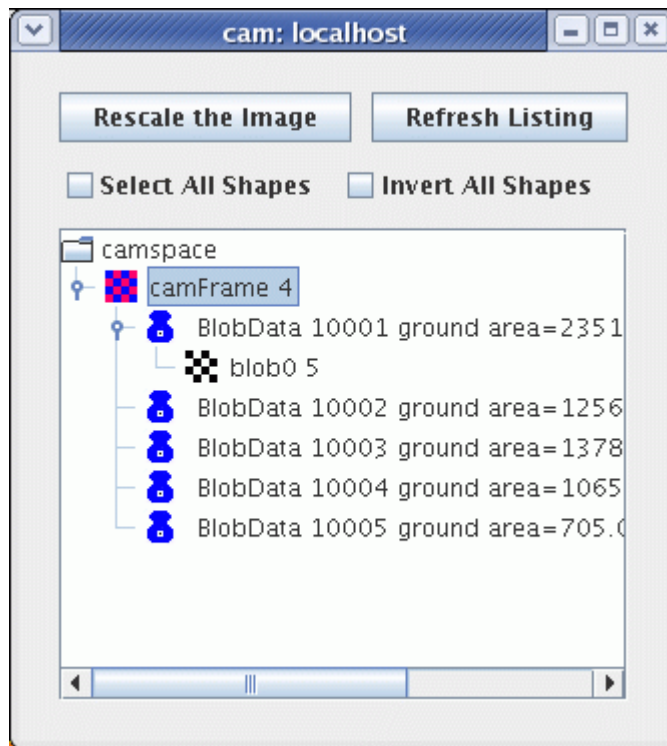


Shapes Are Persistent

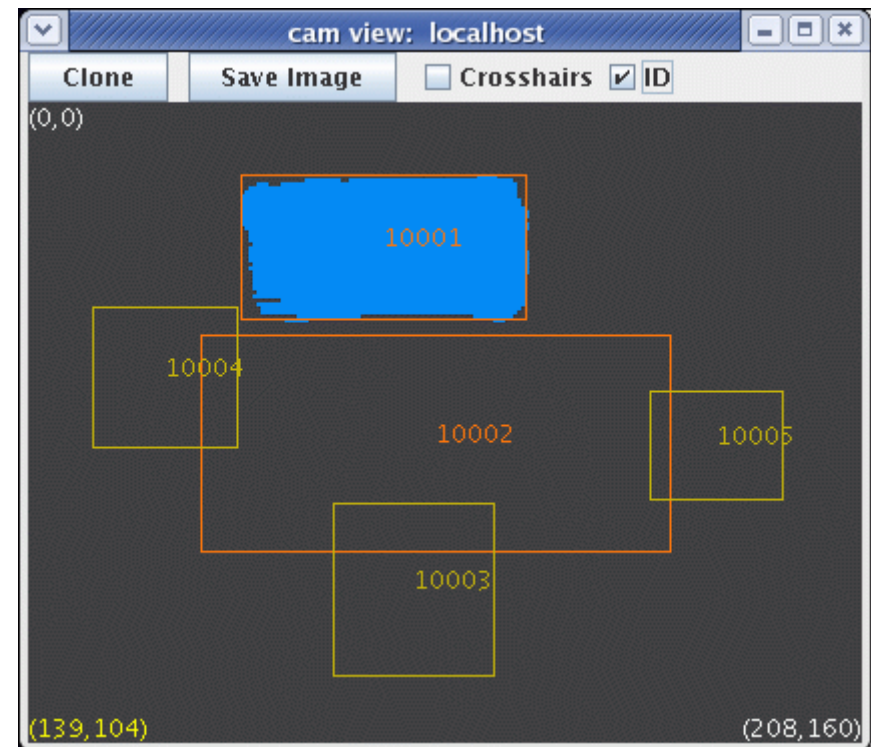
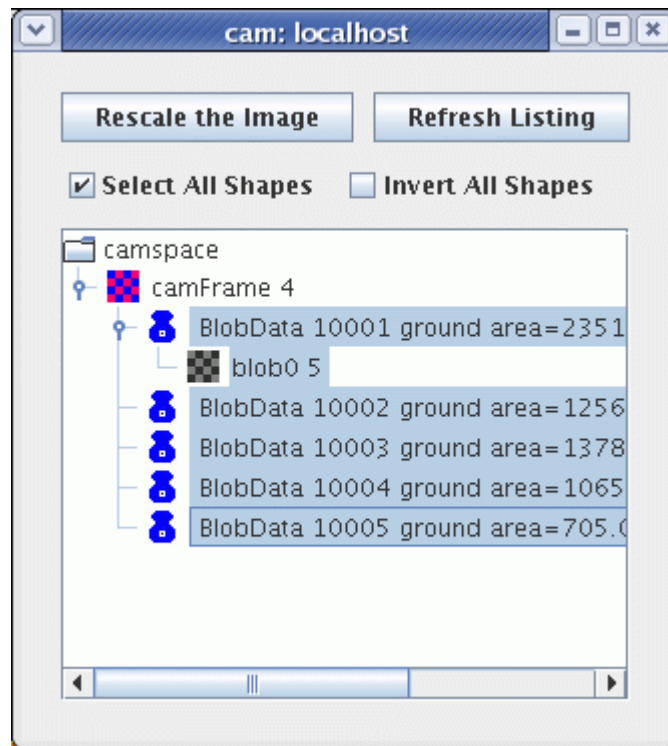
```
$nodeclass Ex3 {  
  
  $nodeclass FindBlobs : MapBuilderNode : doStart {  
    mapreq.addObjectColor(blobDataType, "orange");  
    mapreq.addObjectColor(blobDataType, "yellow");  
  }  
  
  $nodeclass ReportBlobs : doStart {  
    ... (see later slide)  
  }  
  
  $setupmachine{  
    FindBlobs =C=> ReportBlobs  
  }  
  
}
```

The shapes created by FindBlobs will be visible to ReportBlobs because camShS is shared by all state nodes.

Some Orange and Yellow Blobs



Extracted Blob Shapes



SHAPEVEC and SHAPEROOTVEC

- Often we want to work with collections of shapes.
- A “SHAPEVEC” is a vector of shapes of a specific type:

```
std::vector<Shape<BlobData> >
```

This space is
required



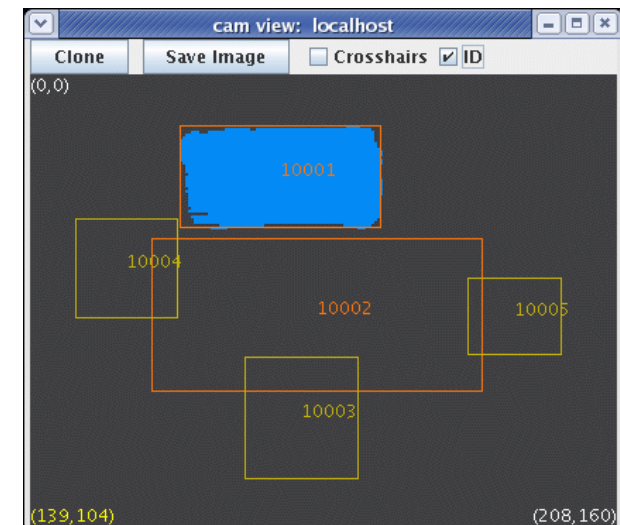
- A “SHAPEROOTVEC” is a vector of generic shapes, useful when we mix shapes of different types:

```
std::vector<ShapeRoot>
```

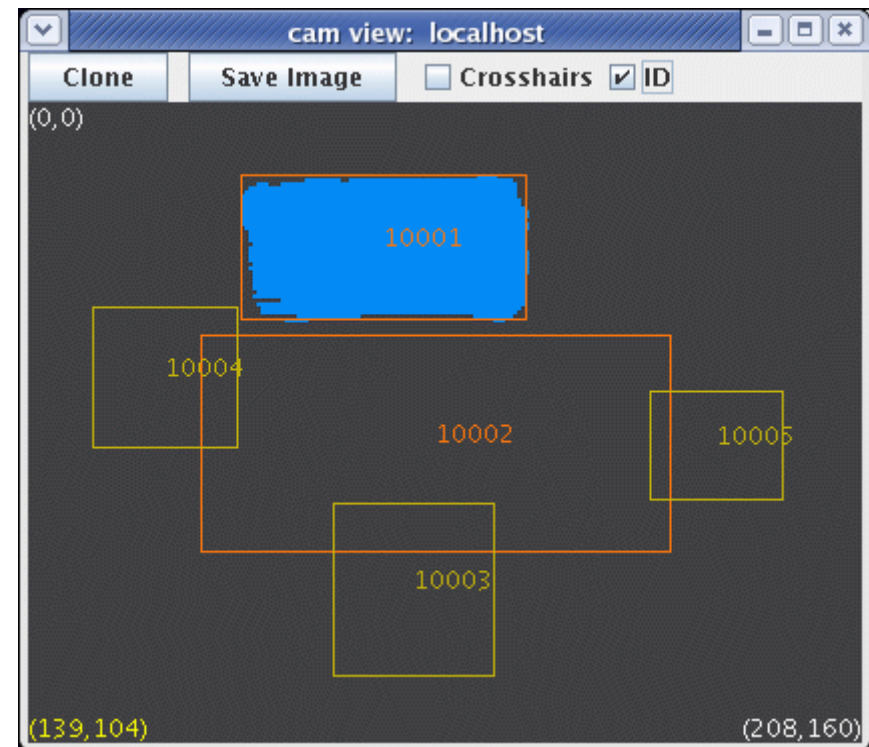
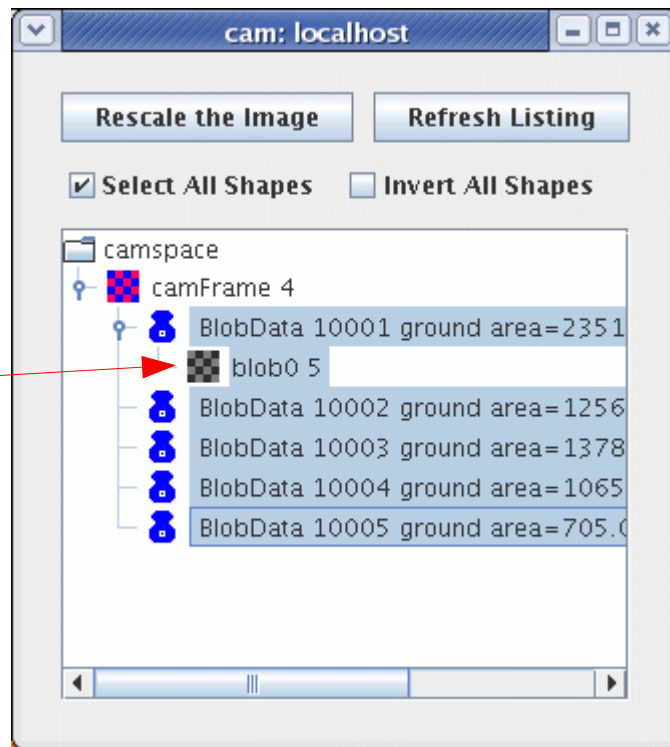
- There are macros for creating and iterating over these vectors:
 - NEW_SHAPEVEC, NEW_SHAPEROOTVEC
 - SHAPEVEC_ITERATE, SHAPEROOTVEC_ITERATE

Vectors of Shapes

```
$nodeclass ReportBlobs : doStart {  
  
    NEW_SHAPEVEC(blob_shapes, BlobData,  
                select_type<BlobData>(camShS));  
  
    if ( blob_shapes.size() > 0 ) {  
        NEW_SKETCH(blob0, bool, blob_shapes[0]->getRendering());  
    }  
  
    SHAPEVEC_ITERATE(blob_shapes, BlobData, myblob) {  
        cout << "Id: " << myblob->getId()  
             << " Color: " << myblob->getColor()  
             << " Area: " << myblob->getArea()  
             << endl;  
    } END_ITERATE;  
  
}
```



Iterating Over Blob Shapes



Id: 10001	Color: [253,119,15]	Area: 2351
Id: 10002	Color: [253,119,15]	Area: 1256
Id: 10003	Color: [193,177,9]	Area: 1378
Id: 10004	Color: [193,177,9]	Area: 1065
Id: 10005	Color: [193,177,9]	Area: 705



Where To Find Stuff

- Sketches and shapes are defined in files in the Tekkotsu/DualCoding directory.
 - LineData.h defines the line class
 - ShapeLine.h defines the smart pointer
 - Everything is in the DualCoding namespace
- MapBuilder is defined in the Tekkotsu/Crew directory.
 - MapBuilderRequest.h defines many options
 - MapBuilderNode.h is used in your state machine
 - MapBuilder.h / MapBuilder.cc

Online Reference Materials

The screenshot shows a web browser window with the URL `tekkotsu.org/dox/index.html`. The page title is "Tekkotsu Reference Documentation". The browser's address bar shows the URL and several icons. The page has a navigation menu at the top with links for "Tekkotsu Homepage", "Demos", "Overview", "Downloads", "Dev. Resources", "Reference", and "Credits". Below the navigation menu are tabs for "Main Page", "Related Pages", "Namespaces", "Classes", "Files", and "Directories". The main content area features a large heading "Tekkotsu Reference Documentation" and a sub-heading "Frames | No Frames". Under the heading "Documentation Contents:", there is a paragraph of text and a section titled "Library Sub-Documentation:" which contains a bulleted list of links: "DualCoding - vision parsing", "Hardware Abstraction Layer - low level device interfacing", "newmat - variable-sized matrix library", and "fmat - fixed-sized (but faster) matrix library". Below this is a section titled "Tekkotsu Documentation:" with a bulleted list of links: "Alphabetical Index - Lists all classes and structs", "Compound List - Gives a short description of each class and struct", "Namespace Members - Lists the global constants, organized by namespaces", "File Members - Lists all of the global variables and macros which aren't in namespaces", and "Related Pages - Links to the todo and bug lists." At the bottom of the page is a section titled "Popular Destinations:". The browser's left sidebar shows a search bar and a list of navigation links under the heading "Tekkotsu", including "Todo List", "Deprecated List", "Class List", "Class Hierarchy", "Class Members", "Namespace List", "Namespace Members", "File List", "Directory Hierarchy", and "File Members".

Search

Tekkotsu

- Todo List
- Deprecated List
- Class List
- Class Hierarchy
- Class Members
- Namespace List
- Namespace Members
- File List
- Directory Hierarchy
- File Members

Tekkotsu Homepage | Demos | Overview | Downloads | Dev. Resources | Reference | Credits

Main Page | Related Pages | Namespaces | Classes | Files | Directories

Tekkotsu Reference Documentation

Frames | No Frames

Documentation Contents:

If you want a more general overview of what this software does and how the pieces fit together, you may want to visit the [overview](#). Don't forget there are also [tutorials](#) available.

Library Sub-Documentation:

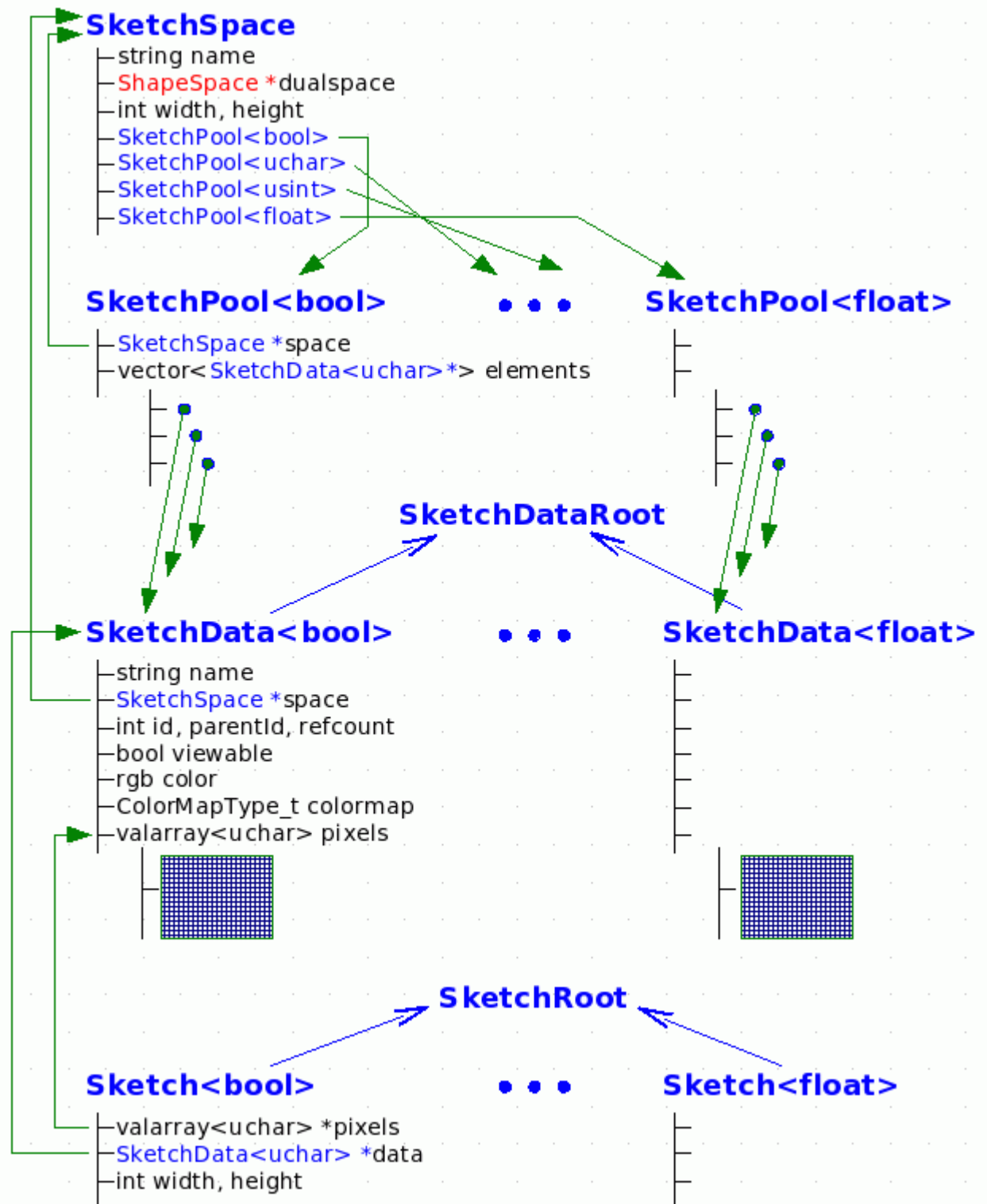
- [DualCoding](#) - vision parsing
- [Hardware Abstraction Layer](#) - low level device interfacing
- [newmat](#) - variable-sized matrix library
- [fmat](#) - fixed-sized (but faster) matrix library

Tekkotsu Documentation:

- [Alphabetical Index](#) - Lists all classes and structs
- [Compound List](#) - Gives a short description of each class and struct
- [Namespace Members](#) - Lists the global constants, organized by namespaces
- [File Members](#) - Lists all of the global variables and macros which aren't in namespaces
- [Related Pages](#) - Links to the [todo](#) and [bug](#) lists.

Popular Destinations:

SketchSpace: A Look Under the Hood



ShapeSpace:

A Look Under the Hood

