Shape Predicates

15-494 Cognitive Robotics
David S. Touretzky & Ethan Tira-Thompson
Carnegie Mellon
Spring 2008
The World is Full of Shapes

- When we extract shapes from camera images, we may get a lot of objects.

- We need ways of selecting and comparing shapes.

- “Find all the orange things.”
  “Find all the lines longer than this line.”

- Tekkotsu provides *shape predicates* for testing shapes. These can be composed to form complex tests.

- To use these, you need to understand C++ functors.
#include <iostream>
using namespace std;

class MyFunctor {
public:
  void operator() () const { cout << "Foo!" << endl; }
};

int main() {
  MyFunctor fluffy;
  
  fluffy();
}
Functors Can Store Values

class BiggerThan {

private:
    int value;

public:
    BiggerThan(int val) : value(val) {}

    bool operator()(int x) const { return x > value; }

};
int main() {
    BiggerThan bigtest(5);
    for (int i = 3; i < 8; i++)
        cout << i << (bigtest(i) ? " passes" : " fails") << endl;
}

3 fails
4 fails
5 fails
6 passes
7 passes
Function Conjunction

class AndBigSmall {
private:
    BiggerThan bigtest;
    SmallerThan smalltest;

public:
    AndBigSmall(BiggerThan b, SmallerThan s) :
        bigtest(b), smalltest(s) {} 

        bool operator() (int x) { return bigtest(x) && smalltest(x); } 
};

int main() {
    AndBigSmall myconj(BiggerThan(0),SmallerThan(100));
    for ( int i = -10; i < 150; i+=40 )
        cout << i << " gives " << myconj(i) << endl;
    -10 gives 0
    30 gives 1
    70 gives 1
    110 gives 0
}
STL functional.h

• The STL (Standard Template Library) provides classes called unary_function and binary_function from which functors can be composed.

```cpp
class BiggerThan : unary_function<int,bool> {
    private:
    int value;
    public:
    BiggerThan(int val) : value(val) {}
    bool operator()(int x) { return x > value; }
};
```

• These user-defined functor classes can then be used with STL functions for searching, etc.

• But they're kind of awkward.
Shape Predicates

• The Shape classes provide their own functor mechanism for defining shape predicates.

• Easier to use than the generic STL.

• Some predicates for common shape tests are built in, e.g.,
  – Comparing the positions of two shapes (left/right or above/below)
  – Comparing the lengths of two lines
  – Comparing line orientations

• New predicates are easy to define.
Shape<LineData> Functors

- Compare the lengths of all the pink lines in the image against that of the third line.

```cpp
NEW_SKETCH(camFrame, uchar, sketchFromSeg());

NEW_SKETCH(pink_stuff, bool,
            visops::colormask(camFrame,"pink");

NEW_SHAPEVEC(lines, LineData,
              LineData::extractLines(pink_stuff));

SHAPEVEC_ITERATE(lines, LineData, ln)
    if ( LineData::LengthLessThan()(ln,lines[2]) )
        cout << "Shorter: " << ln->getId() << endl;
    else
        cout << "Longer: " << ln->getId() << endl;
END_ITERATE;
```
Class-specific shape predicates are defined with the respective shape, e.g., in LineData.h and LineData.cc.

In LineData.h:

class LengthLessThan : public BinaryShapePred<LineData> {
    public:
    bool operator() (const Shape<LineData> &ln1, 
                    const Shape<LineData> &ln2) const;
};

In LineData.cc:

void LineData::LengthLessThan::operator()
    (const Shape<LineData> &line1, 
     const Shape<LineData> &line2) const {
    return line1->getLength() < line2->getLength(); }
Generic Shape Predicates

- Some predicates work for shapes of any type. They are defined on class ShapeRoot. Example: IsColor.

    NEW_SHAPEVEC(blobs, BlobData, BlobData::extractBlobs(camFrame,50));

    IsColor orangetest("orange");

    SHAPEVEC_ITERATE(blobs, BlobData, b) 
    if ( orangetest(b) )
        cout << "Orange: " << b->getId() << endl;
    else
        cout << "Not orange: " << b->getId() << endl;
    END_ITERATE;
Subclasses of BaseData:

- ShapeSpace
  - string name
  - SketchSpace *dualspace
  - ReferenceFrameType_t refFrameType
  - int num_shapes, first_free, id_counter
  - vector<ShapeRoot> allShapesCache
  - vector<BaseData*> data_ptrs

- LineData
  - ShapeSpace* space
  - string name
  - ShapeType_t type
  - int id, parentld, refcount
  - bool viewable
  - rgb color
  - Sketch<bool> *rendering_sketch
  - Endpoint end1pt, end2pt

- BlobData
  - ShapeSpace* space
  - int id, indx
  - vector<BlobData::run> runvec

Subclasses of ShapeRoot:

- Shape<LineData>
- Shape<BlobData>
Generic IsColor Predicate

class IsColor : public UnaryShapeRootPred {
private:
    rgb color;

public:
    IsColor(rgb col) : UnaryShapeRootPred(), color(col) {}
    IsColor(std::string const &colorname) :
        UnaryShapeRootPred(),
        color(ProjectInterface::getColorRGB(colorname)) {}

    bool operator() (const ShapeRoot &shape) const {
        return shape->getColor() == color;
    }
};

Note: the colorname string is looked up once, by the constructor, and the result is stored in the private variable color. When the functor is invoked on a ShapeRoot, no lookup is necessary.
IsLeftOf / IsLeftOfThis

• IsLeftOf()
  
  - This is a BinaryShapeRootPred that requires two arguments, and compares their centroids:

    IsLeftOf() (line2,blob6)

• IsLeftOfThis(x)
  
  - This is a UnaryShapeRootPred that requires one argument:

    IsLeftofThis(line2) (blob6)

constructor  argument
Using IsLeftOfThis

• An instance of IsLeftOfThis stores a ShapeRoot inside it, and uses it for comparison tests.

```cpp
IsLeftOfThis mytest(lines[4]);

SHAPEVEC_ITERATE(lines, LineData, ln)
    if ( mytest(ln) )
        cout << "This is left of me: " << ln->getId() << endl;
END_ITERATE;
```
Built-In Shape Predicates

ShapeRoot:
- IsColor
- IsType
- IsName
- IsLeftOf / IsRightOf
- IsAbove / IsBelow
- IsLeftOfThis ...
- IsAboveThis ...

Shape<LineData>:
- LengthLessThan
- IsHorizontal
- IsVertical
- ParallelTest
- PerpendicularTest
- ColinearTest
AndPred / OrPred

- Because shape predicates are classes, we can compose them using the functors AndPred and OrPred.

```cpp
SHAPEVEC_ITERATE(lines, LineData, ln)
    if ( AndPred(IsColor("pink"),
                 IsLeftOfThis(lines[3])) (ln) )
        cout << "winner: " << ln->getId() << endl;
    else
        cout << "loser: " << ln->getId() << endl;
END_ITERATE;
```

- We are composing two unary predicates, so the result is also a unary predicate: it takes one argument.
**Vectors of ShapeRoots**

- camShS.allShapes() returns all the shapes in the shape space, as a vector<ShapeRoot>.
- camShS will be automatically coerced to vector<ShapeRoot> by an implicit call to allShapes()
- Use SHAPEROOTVEC_ITERATE(vec,var) to iterate:

  ```cpp
  SHAPEROOTVEC_ITERATE(camShS, s)
  if ( OrPred(IsType(blobDataType),
              IsType(lineDataType))(s) )
    cout << "Is blob or line: " << s->getId() << endl;
  END_ITERATE;
  ```

- Shape type constants like `blobDataType` are defined in `ShapeTypes.h`
Mirroring STL Search Functions

• The STL provides a collection of functions for searching through a vector using either a binary comparison predicate or a unary test predicate.

• Tekkotsu provides similar functions for shape predicates:
  - find_if, subset, max_element, stable_sort, remove_copy_if

• There are also some new functions unique to shapes:
  - find_shape, select_type
Filtering Shapes

• Find the first blob:

NEW SHAPE(blob0, BlobData, find_if<BlobData>(camShS));

• camShS is treated as shorthand for camShS.allShapes()
• If no blobs found, an invalid Shape is returned

• Find all the blobs:

NEW SHAPE_VEC(all_blobs, BlobData,
   select_type<BlobData>(camShS));
More Filtering and Searching

- Find all the orange blobs:

  ```
  NEW_SHAPEVEC(orange_blobs, BlobData,
                subset(all_blobs, IsColor("orange")))
  ```

- Find the longest line:

  ```
  NEW_SHAPE(longest, LineData,
            max_element(lines,
                        LineData::LengthLessThan()))
  ```

- Test is “less than”, but max_element returns `longest`. 
Implementing max_element

// from DualCoding/ShapeFuns.h

template<class T, typename ComparisonType>
Shape<T> max_element(const vector<Shape<T> > &vec,
                      ComparisonType comp) {

    typename vector<Shape<T> >::const_iterator result =
        max_element(vec.begin(), vec.end(), comp);

    if (result != vec.end())
        return *result;
    else
        return Shape<T>();

}
Negating a Predicate

- Use not1(p) to negate a unary predicate:

```cpp
NEW_SHAPEROOTVEC(non_orange,
    subset(camShS, not1(IsColor("orange"))));
```

- Use not2(p) to negate a binary (comparison) predicate:

```cpp
NEW_SHAPEVEC(shortlines, LineData,
    stable_sort(lines, not2(LineData::LengthLessThan())));
```
SHAPEVEC_ITERATE(lines, LineData, ln)
    do_something_with(ln);
END_ITERATE;

Expands into:

for ( vector<Shape<LineData> >::iterator ln_it = lines.begin();
    ln_it != lines.end(); ln_it++ ) {
    Shape<LineData> &ln = *ln_it;
    do_something_with(ln);
};
Nested Iteration

NEW_SHAPEVEC(lines, LineData, select_type<LineData>(camShS));
lines = stable_sort(lines, not2(LineData::LengthLessThan()));

SHAPEVEC_ITERATE(lines, LineData, ln1)
  SHAPENEXT_ITERATE(lines, LineData, ln1, ln2)
    if ( LineData::ParallelTest()(ln1, ln2) )
      cout << ln1 << " parallel to " << ln2 << endl;
    if ( LineData::PerpendicularTest()(ln1, ln2) )
      cout << ln1 << " perpendicular to " << ln2 << endl;
    if ( LineData::ColinearTest()(ln1, ln2) )
      cout << ln1 << " colinear with " << ln2 << endl;
  END_ITERATE;
END_ITERATE;

Shape<LineData>(id=10002, indx=1) perpendicular to
Shape<LineData>(id=10005, indx=4)
... etc.