Constructive Solid Geometry in Two Dimensions

Prof. Jeremy Roberts

10/5/2015

Outline

Key Ideas

1. **Geometry** is $\cup$ of solid bodies, which are areas in 2-D
2. **Solid bodies** defined by **surfaces**, or $\cup$ or $\cap$ of other bodies
3. Surfaces defined **implicitly**, or by $\cup$ or $\cap$ of other surfaces

Geometry applications are natural targets for use of object-oriented programming. By creating your own minimal, 2-D CSG engine, you'll quickly discover the power of classes.

Implicit Surfaces

Any two-dimensional surface can be defined implicitly as

$$f(x, y) = 0 \tag{1}$$

By convention, the point (x, y) is

1. inside the surface if $f(x, y) < 0$
2. outside the surface if $f(x, y) > 0$
3. on the surface if $f(x, y) = 0$

Therefore, the surface consists of those points such that $f(x, y) = 0$.

Often, we want to find those points (e.g., to plot the surface) or the intersections of the surface with a ray, another surface, etc. Both tasks require solving the potentially nonlinear equation $f(x, y) = 0$.

Quadratic Surfaces

A 2-D quadratic (or second-order) surface is defined implicitly as

$$f(x, y) = Ax^2 + By^2 + Cxy + Dx + Ey + F\,. \tag{2}$$

Let

$$\mathbf{r} = [x, y, 1]^{\mathsf{T}} \quad \text{and} \quad \mathbf{M} = \begin{bmatrix} 2A & C & D \\ C & 2B & E \\ D & E & 2F\,. \end{bmatrix}$$

Then,

$$f(x, y) = \frac{1}{2}\mathbf{r}^{\mathsf{T}}\mathbf{M}\mathbf{r}\,. \tag{3}$$

Prove this to yourself!

Ray/Surface Intersections

Common problem: *where does a ray intersect a surface?*

Let a ray $\mathbf{r}$ be defined as

$$\mathbf{r} = \mathbf{r}_0 + t\mathbf{d} \tag{4}$$

where $\mathbf{r}_0$ is some starting point, $\mathbf{d}$ is some direction (so $|\mathbf{d}| = 1$), and t is the distance from the starting point along the direction.

Substitute this ray into Eq. (3) to find where the ray intersects the surface (and, hence, $f = 0$). The result is (**show this!**)

$$t^2 \overbrace{\mathbf{d}^\mathsf{T}\mathbf{M}\mathbf{d}}^{a} + 2t \overbrace{\mathbf{r_0}^\mathsf{T}\mathbf{M}\mathbf{d}}^{b} + \overbrace{\mathbf{r_0}^\mathsf{T}\mathbf{M}\mathbf{r_0}}^{c} = 0\,, \tag{5}$$

which is quadratic in t. If

$b^2 > 4ac$	there are two intersections
$b^2 = 4ac$	there is one intersection (tangent)
$b^2 < 4ac$	there are no intersections
$a = 0$	the surface is linear

Proposed Structure

- `class Point(object)`
- `class Ray(object)`
- `class Node(object)`
  - `class Surface(Node)`
    - `class QuadraticSurface(Surface)`
  - `class Operator(Node)`
    - `class Primitive(Operator)`
    - `class Union(Operator)`
    - `class Intersection(Operator)`
- `class Region(object)`
- `class Geometry(object)`

Point

```
class Point(object) :

    def __init__(self, x, y) :
        self.x = x
        self.y = y

    def __str__(self) :
        return " Point(%.6f, %.6f) " % (self.x, self.y)
```

Ray

```
class Ray(object) :

    def __init__(self, origin, direction) :
        """ Initialize a Ray with a given origin and direction.

        Arguments:
            origin: Point
            direction: Point
        Return:
            none
        """
        self.origin = origin
        # ensure the direction is normalized to unity
        norm = np.sqrt(direction.x**2 + direction.y**2)
        x, y = direction.x/norm, direction.y/norm
        self.direction = Point(x, y)

    def __str__(self) :
        """ Return string representation of Ray.
        """
        return "Ray: r_0(%10.6f, %10.6f), d(%.6f %.6f) " % \
               (self.origin.x, self.origin.y,
                self.direction.x, self.direction.y)
```

Surface

```
class Surface(Node) :

    def f(p) :
        """Function that implicitly defines the surface."""
        raise NotImplementedError
```

Node

```
class Node(object) :

    def contains(self, p) :
        """Does the node contain the point?"""
        raise NotImplementedError

    def intersections(self, r) :
        """Where does the node intersect the ray?"""
        raise NotImplementedError
```

Primitive

```
class Primitive(Node) :

    def __init__(self, surface, sense) :
        """ Define a node consisting of a directed surface.

        Here, sense indicates "into" or "outof" the surface.

        Arguments:
            surface : Surface (or derived variants)
            sense : bool
        """

        self.surface, self.sense = surface, sense

    def contains(self, p) :
        return (self.surface.f(p) < 0) == self.sense

    def intersections(self, r) :
        return self.surface.intersections(r)
```

Operator

```
class Operator(Node) :

    def __init__(self, L, R) :
        """ Create an operation between two nodes.
        """
        self.L, self.R = L, R

    def contains(self, p) :
        raise NotImplementedError

    def intersections(self, r) :
        """ Return a list (maybe empty) of intersection points.
        """
        pointsL = self.L.intersections(r)
        pointsR = self.R.intersections(r)
        # return the concatenated result
        return pointsL + pointsR
```

Region

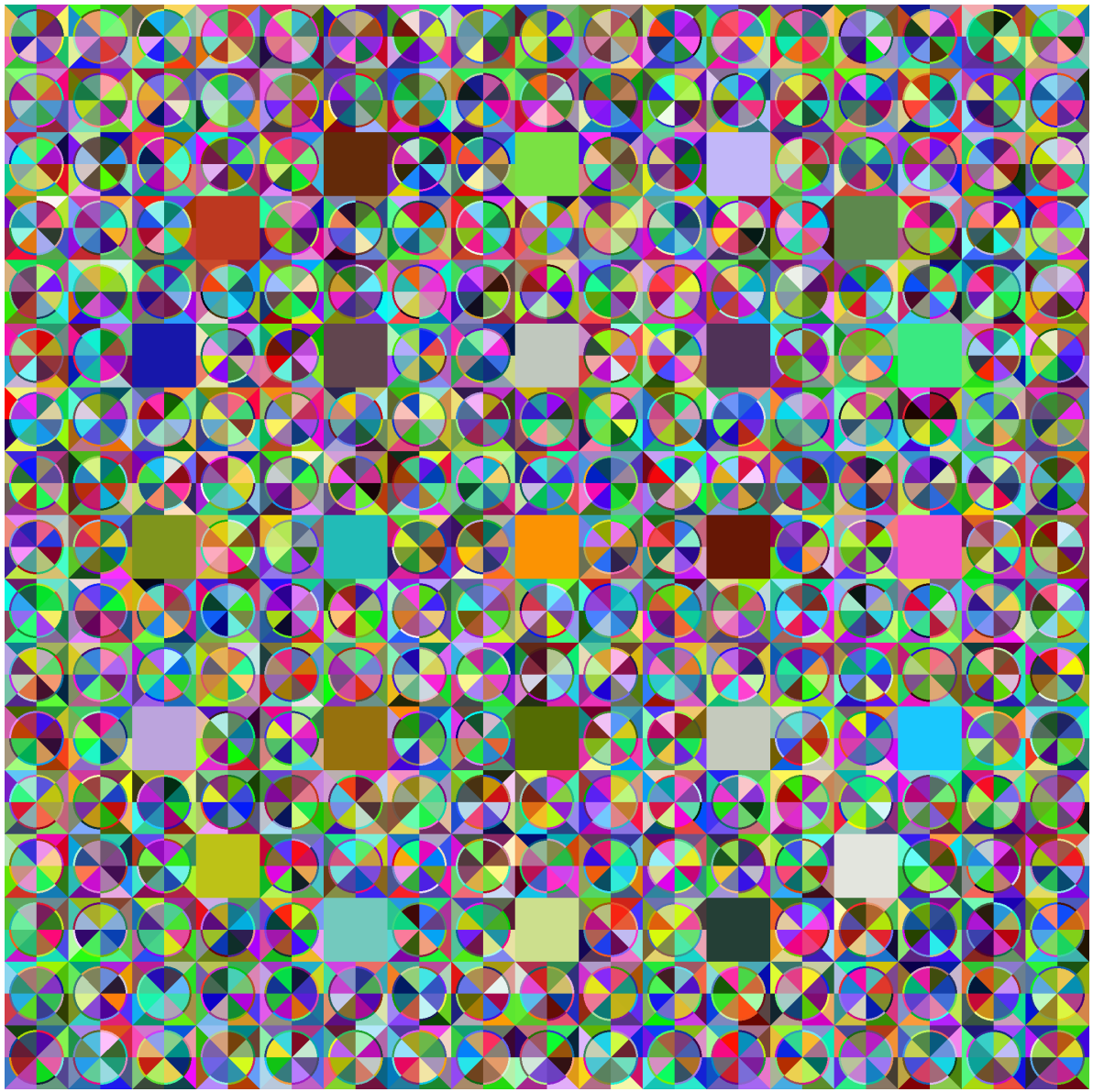
```
class Region(object) :

    def __init__(self) :
        self.node = None

    def append(self, node=None, surface=None,
               operation="U", sense=False) :
        assert((node and not surface) or (surface and not node))
        if isinstance(surface, Surface) :
            node = Primitive(surface, sense)
        if self.node is None :
            self.node = node
        else :
            O = Union if operation == "U" else Intersection
            self.node = O(self.node, node)

    def intersections(self, r) :
        pass
```

Geometry

```
class Geometry(object) :
    noregion = -1

    def __init__(self,  xmin, xmax, ymin, ymax) :
        self.xmin, self.xmax = xmin, xmax
        self.ymin, self.ymax = ymin, ymax
        self.regions = []

    def add_region(self, r) :
        self.regions.append(r)

    def find_region(self, p) :
        """ Find the region that contains the point.  If none
        is found, return Geometry.noregion.

        Arguments:
            p : Point
        Returns:
            i : int
        """
        pass

    def plot(self, xmin, xmax, nx, ymin, ymax, ny) :
        # see the template file; discussion to follow
```

How to Plot?

Simple way: **create x-y grid of pixels.**



Example

```
c0 = Circle(r=0.5,x0=2,y0=0)
c1 = Circle(r=0.5,x0=-2, y0=0)
R0 = Region()
R0.append(surface=c0, sense=True)
R1 = Region()
R1.append(surface=c1, sense=True)
geo = Geometry()
geo.add_region(R0)
geo.add_region(R1)
geo.plot(-5, 0, 100, -5, 0, 100)
```

Example