Computer Science 112
Art & Science of Computer Graphics
The College of Saint Rose
Spring 2016

Topic Notes: Cameras

Image Sizes

Before we discuss cameras, a bit more about Ambrosia images.

We have seen at least three messages that we can send to Ambrosia's builtin `image` object:

- background color

  ```
  image.background(blue)
  ```

- image quality

  ```
  image.quality(10)
  ```

  This sets the image quality to 10. The default is 8. Lower quality values render images less accurately but quickly. This may be useful when developing complex models that take a long time to render. Higher quality images are more accurate at the expense of increased rendering time.

- dimensions of the generated image

  ```
  image.dimensions(1024, 768)
  ```

  This sets the number of *pixels* (short for *picture elements*) in the generated image to have 1024 columns and 768 rows.

  Each pixel can store only a single color value. More pixels means a more detailed image when we view it.

There are other ways that we can change the dimensions.

First, note that the ratio of the width to the height of the image is called the *aspect ratio* of the image.

Often, we choose an aspect ratio based on the size of our display. A standard television set has an aspect ratio of $\frac{4}{3}$. A widescreen television set has an aspect ratio of $\frac{16}{9}$.

The default image size is 640x480, giving an aspect ratio of $\frac{4}{3}$.

Other messages we can send to the `image` that affect the size of the generated image.

- `width` – set the width of the image, changing the height as appropriate to maintain the current aspect ratio

  ```
  image.width(800)
  ```

- `height` – sets the height, again maintaining aspect ratio

  ```
  image.height(600)
  ```

- `aspectRatio` – set the aspect ratio by adjusting the height and keeping the width fixed

  ```
  image.aspectRatio(2)
  ```

  The above, if applied to an image with the default 640x480 dimensions, will result in an image of size 640x320 being generated.

  Note that an aspect ratio below 1 results in a "portrait" image, 1 results in a square image, and above 1 results in a "landscape" image.

There is one more message we can send to the `image`, and that sets the level of *antialiasing*.

```
image.antiAlias(.3)
```

Antialiasing is used to "smooth" parts of the image that will look jagged or "pixelated" because sharp curved surfaces are being represented by pixels that are either part of the surface or not. Antialiasing makes the image look smoother by doing some averaging of pixels near the edge.

The values range from 0 for no antialiasing to 1 for maximum antialiasing. The default value is 0.5.

On the Wiki: AntiAliasExample

The `Camera` Class

We have been using Ambrosia's predefined `camera` object, which is an instance of the `Camera` class, all semester.

We know of some of the parameters we can set on this camera:

- position – where is the camera in our universe

  ```
  camera.pos([100, 100, -400])
  ```

This places the camera at (100,100,-400).

The default position of Ambrosia's predefined `camera` object is (0,0,-500).

- center of interest – where is the camera pointed

```
camera.COI([100, 100, 0])
```

This points our camera at the point (100,100,0).

The default center of interest for Ambrosia's predefined `camera` object is the origin.

We can also set the orientation of the camera, by telling it the direction of "up":

```
camera.up([1, 0, 0])
```

This tells the camera that the "up" direction should be pointing along the positive x-axis.

By default, the camera's up is defined along the positive y-axis (0,1,0).

On the Wiki: CameraUp

Camera View Angles

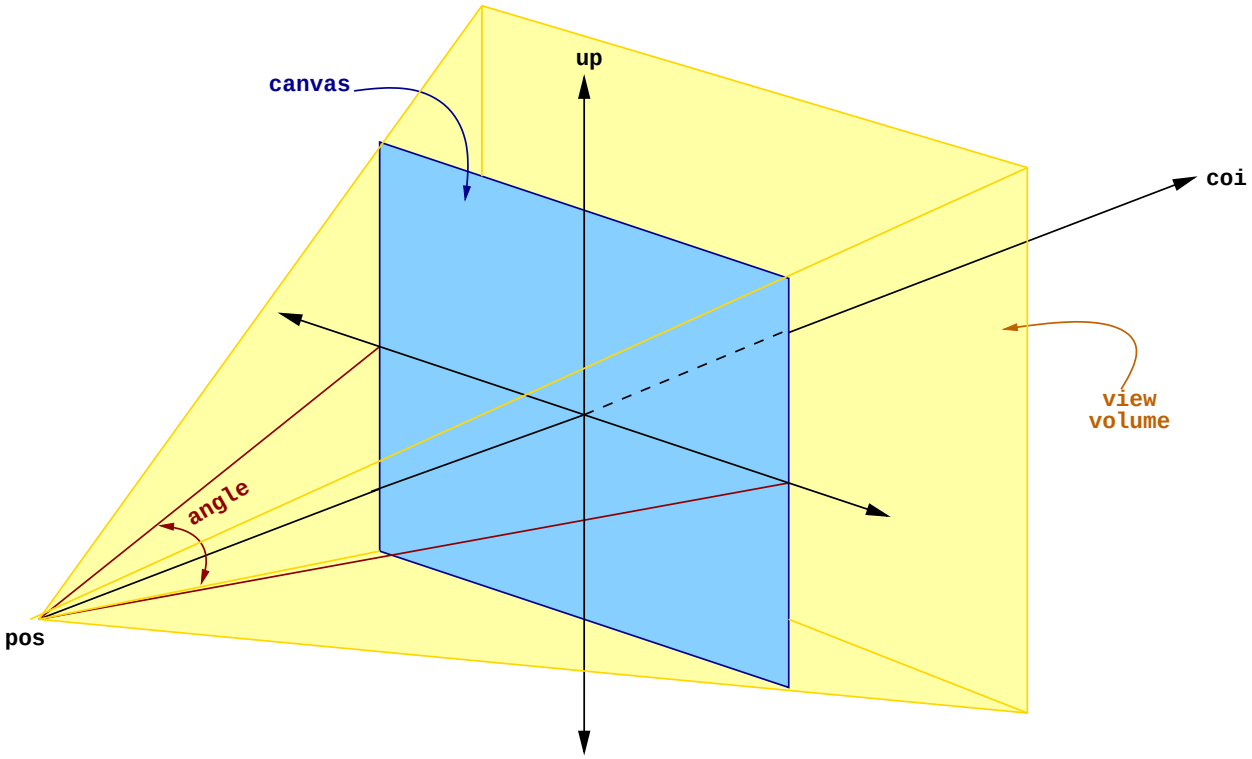
The next camera attribute we can set is called the `angle`, but before we can see what it means, let's consider what a camera needs to do to generate an image from a scene.

Our camera is at a given point in space and it is looking at another point in space. We also can define the "up" direction to make sure the camera orientation is what we want.

The question then becomes, how to we decide which part of the scene is visible in our image? Of course, it is those parts of the scene that are "in the same direction" as our center of interest.

We can think of the image we generate as a window out into the scene. The distance from the camera position to the window determines how much of the scene will be visible in our image. If the camera is very close to this window, we get a wide view of the scene. If we are very far away, we get a much narrower view.

In Ambrosia, we adjust this by setting a camera attribute called the *camera lens angle*, as shown in this image:



The yellow shaded pyramid in this figure (which extends back infintely) is the *view volume*, and the objects we see are everything in the view volume that is not occluded by an object between it and the camera position (called its *viewpoint*). The blue rectangle, labeled the "canvas", is the intersection of the view volume with the *image plane*.

The smaller the angle, the further the camera is from the image plane, and the narrower our view to the world. A larger angle, and we have a wider view.

```
camera.angle(45)
```

- The default `angle` for Ambrosia's camera is 53 degrees
- Anything in the 50-60 degree range is considered "normal"
- 5-30 degrees would be considered "telephoto"
- 90 degrees or higher would be "wide angle"

On the Wiki: CameraLensAngles

Orthographic Cameras

The default `camera` operates in "perspective" mode – meaning objects further away in the scene look smaller.

We can change this behavior by sending the `camera` the message:

```
camera.orthographic()
```

This switches to an *orthographic view* instead of the *perspective view*.

While this may not seem useful for realistic views, it may be helpful when trying to determine relative sizes of objects in a scene.

The camera can be returned to its default perspective mode with:

```
camera.perspective()
```

On the Wiki: OrthographicVsPerspective

Isometric Camera

As you can see in the above example, Ambrosia also provides a special camera that you may find useful when debugging your models. The `IsometricCamera` takes four views of your scene: from the top, from the front, from the right side, and from an angle above, to the right, and out.

```
isoCamera = IsometricCamera()
isoCamera.subject(scene)
isoCamera.shoot()
```

By default, the `IsometricCamera` uses an orthographic view. This can be changed with the `(perspective)` message. Note that when you construct a new `IsometricCamera` its `subject` does not default to the `scene`. We have to send it that message explicitly.