

4 - Camera Control

Overview

- “MouseLook” Mode and Cursors
- 6 DoF vs. Constrained Cameras
- 1st-person vs. 3rd-person Cameras
- Chase Cameras
- Heads-Up Displays (HUDs)
- Viewports / Split Screen

2

“Mouse-Look” Mode*

“Mouse-look” == using mouse to control
camera orientation (introduced in “Quake” c. 1996)

Two methods of obtaining mouse moves:

- Input Manager axis devices
- Window Manager mouse listener routines
AWT MouseMoved, MouseDragged, etc.

* also known as “Free-Look” mode

3

“Mouse-Look” (continued)

Challenges with WindowManager mouse control:

- Mouse stops at screen edge
- Player can't move camera beyond that limit







Java solution:

- *recenter* mouse after each move
- Java Robot class can be used for this
- Can also consider altering (or removing) the cursor

4

Setting Mouse Cursors

Some Java pre-defined cursors:

-  Cursor.DEFAULT_CURSOR
-  Cursor.CROSSHAIR_CURSOR
-  Cursor.TEXT_CURSOR
-  Cursor.WAIT_CURSOR
-  Cursor.HAND_CURSOR
-  Cursor.MOVE_CURSOR

Obtaining a cursor:

```
Cursor waitCursor =  
    Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR);
```

Changing the current cursor in RAGE:

```
rendersystem.getCanvas().setCursor(waitCursor);
```

5

Setting Mouse Cursors (cont.)

Defining your own custom cursor

```
Cursor cursor = Toolkit.getDefaultToolkit().  
    createCustomCursor(Image i, Point hotSpot, String name);
```

Example:

```
Image pencilImage =  
    new ImageIcon("images/pencil.gif").getImage();  
Cursor pencilCursor =  
    Toolkit.getDefaultToolkit().  
        createCustomCursor(pencilImage, new Point(0,0),  
                            "PencilCursor");
```

(note: “Toolkit” is part of Java AWT)

6

Setting Mouse Cursors (cont.)

Invisible cursors

- Not predefined in Java
- Can be created using an "undefined image"

```
Toolkit tk = Toolkit.getDefaultToolkit();
Cursor invisibleCursor =
    tk.createCustomCursor(tk.getImage(""),
        new Point(), "InvisibleCursor");
rendersystem.getCanvas().setCursor(invisibleCursor);
```

7

Unconstrained (6 DoF) Cameras

Consider the following camera sequence:

```
Rotate(90, Y)
Rotate(90, X)
Rotate(-90, Y)
Rotate(-90, X)
```

Does it put the camera back to initial state?

Why not?

The same effect creeps into camera control in small (but *cumulative*) amounts with small rotations...

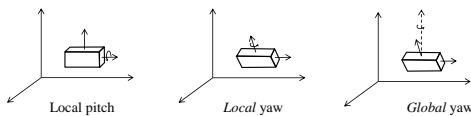
8

Constrained Cameras

6 DoF "flight": pitch + yaw introduces *roll*

- Appropriate for "flight simulators" or "spaceships"
- Not appropriate for ground-based FPS games (*looking around* shouldn't cause *roll*)

Control by using local pitch, but global yaw



9

Transformation matrix for *Local Yaw*:

```
Matrix4f.createRotationFrom(rotationAngleAmt,
    camera.getUp());
```

Transformation matrix for *Global Yaw*:

```
Matrix4f.createRotationFrom(rotationAngleAmt,
    Vector3f.createFrom(0, 1, 0));
```

10

1P vs. 3P Cameras

First-Person (1P) Cameras:

- Located at the player's "point of view"
- Player's loc/dir changed by manipulating *camera*

Gaming characteristics of 1P:

- Good for "local environment" feedback sounds
Heartbeat, breathing, footsteps, weapon sounds
- Provides limited view of surroundings
Things can "sneak up" (good for building *suspense*)
- Easier to "aim" in shooting games

11

Types of 3P Cameras

Bird's-eye ("2 1/2 D" perspective)

- Fixed camera looking down on a (mostly) 2D world
- Player avatar is independent of camera
- Some games have no avatar (e.g., building games such as *SimCity*, or Real-Time Strategy games such as *Age of Empires*)

Examples:



Sim City 2000



Starcraft



League of Legends

12

Types of 3P Cameras (cont.)

Chase (also called *tracking*)

- Camera follows avatar, maintains constant relative view ("over-the-shoulder"; "behind-the-back")
- Camera typically on "springs" to reduce jerkiness

Examples:



Mario Kart



Mario Kart 64 Battle Mode

13

Types of 3P Cameras (cont.)

"Targeted" (also called *orbit*)

- Camera always looks at avatar
- Camera can be independently controlled in various ways (orbit, zoom)
- In some games, zooming all the way in puts camera in 1P mode.

Example: World of Warcraft



Camera behind avatar



Camera orbited around avatar

14

Building a Targeted 3P Camera

Camera characteristics:

- Location: typically starts "above" and "behind" avatar
- Focal point: usually directed *at (or slightly above)* avatar

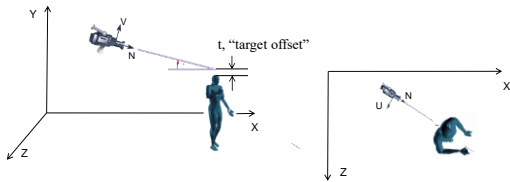


Image credit: www.gamasutra.com

15

3P Camera Positioning

- Orbit camera position defined in *spherical coordinates*:

Azimuth θ , altitude (elevation) Φ , radius (distance) R

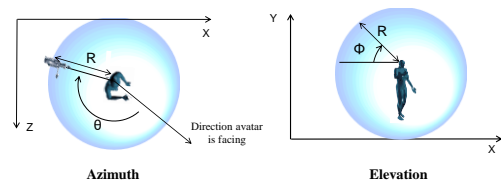
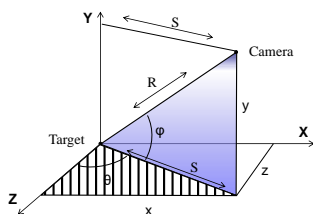


Image credit: www.gamasutra.com

16

Computing Spherical Position



$$S = \sqrt{x^2 + z^2} = R \cos(\varphi)$$

$$R = \sqrt{S^2 + y^2} = \sqrt{x^2 + y^2 + z^2}$$

$$x = S \sin(\theta) = R \cos(\varphi) \sin(\theta)$$

$$y = R \sin(\varphi)$$

$$z = S \cos(\theta) = R \cos(\varphi) \cos(\theta)$$

Note that here x , y , & z are *relative to target location*; need to add target location to get camera world position

17

Targeted Camera Avatar Control

Typical controls:

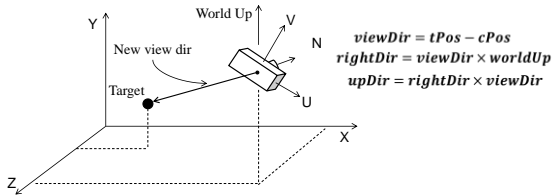
- ASWD moves **avatar** (3P camera "follows")
- Mouse X/Y controls camera *azimuth* and *elevation*
- Mouse *wheel* controls *distance* (zoom)
- Avatar may *rotate* with camera rotation (e.g. when right mouse button is down)

18

Computing Look-At

Given: a camera *position* and *orientation*
(U,V,N axis directions)

Needed: function `lookAt(target, worldUp)`



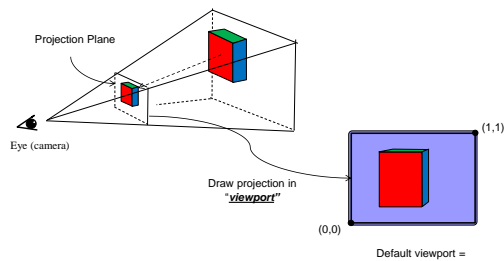
19

Multi-Player "Split-screen"



20

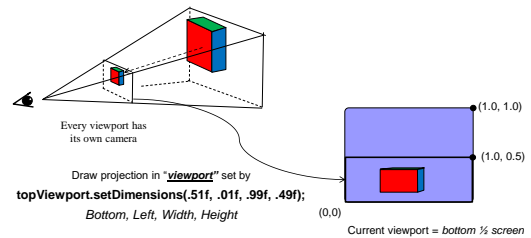
Viewports



21

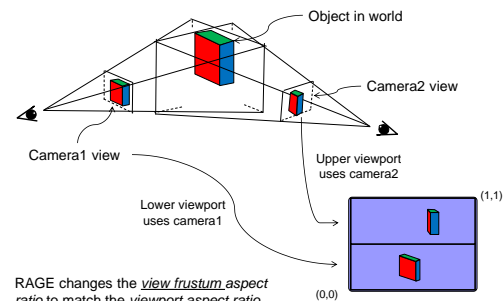
Changing the Viewport

```
rw.setViewport(0).setCamera(camera1);
```



22

Multiple Cameras



23

Multiple Cameras (cont.)

Game maintains a collection of viewports,
and a collection of cameras,
and a collection of HUD objects.

setupCamera() and **setupScene()**

- Creates cameras (one per player)
- Associates input controls with different cameras
- Defines HUD items for each viewport

render() draws *each camera's* view, in that camera's viewport, in perspective projection.

HUDs are drawn in orthographic projection.

24