Advanced Camera Control

IMGD 4000


“An ideal virtual camera system, regardless of genre, is notable by the lack of attention given to it by the viewer”

From Introduction to:

God of War 2 trailer https://www.youtube.com/watch?v=GjYbK_-w9pM

Note: if you don’t notice camera, then it is working well!

Camera Objectives

- Flexible and designer driven
  - Allow game designer to provide player experience from variety of perspectives
- Smooth
  - No jarring transitions
- Not require player intervention
  - Player should not have to manually adjust camera to see game
- No collision
  - Designer must constrain so doesn’t go through walls

Overview

- **Zoning** – deals with use of spatial database to select “right” camera
- **Dynamics** – calculations for a single, dynamic camera
- **Blending** – smooth out transitions between cameras
- **Rails** – constraining camera to path

Any one of these techniques counts as optional tech (although you probably want to use some of them together).
Zoning: Objectives

- Have multiple stationary cameras
  - Cameras in fixed location
- Chosen by player position
  - Active camera is based on where player is
- Design so that cameras can "cover" where player is
- Switch automatically to right camera

Zoning: Design

Select camera from database based on player zone location.
If move border, will "toggle" between cameras.

Non-overlapping zones. Switch to camera when player ENTERS zone.
Provides hysteresis.
But may not "cover" all areas well.

Alternative is overlapping zones.
When enter overlap, switch to new camera.
Each frame, query spatial DB and get cameras. Assume unordered (don’t want assumption about underlying db).

Camera C
Camera A
ACB

A B C
Zoning : Implementation

If simply switch to new camera, will toggle between A and C every frame.

Submission List
– List of all cameras that were submitted last frame.
– Used to distinguish newly submitted cameras from old ones
– New cameras inserted at top
  • So, effectively sorted by age

Query Result
Submission List
A
B
C

In this example, player moves from A to C, to B.
Zoning : Implementation

Camera A
Priority 1

Provide priority for more important (higher priority) camera when overlapped.

Camera B
Priority 2

Zoning Implementation

• Submission List
  – Insert and delete entries to match query results
  – Unless query result was empty
  – Sorted by priority
  – Then by age
  – Top entry is active camera

Outline

• Zoning (done)
• Dynamics (next)
• Blending
• Rails
Dynamics: Objectives

- Control following three properties of player avatar as it appears on screen:
  - Position – where on the screen avatar appears (e.g., center? bottom right?)
  - Size – how big avatar is (e.g., takes up full screen, takes up tiny portion)
  - Angle – what representation avatar has (e.g., profile, top-down)

Dynamics: Design

- Define "safe zone" (rectangle) on screen where player avatar will be.
- Make these resolution independent (represent screen, not pixels).
- Note, can be "point" if player always there.
- When avatar outside of zone, move camera. How?

Dynamics: Design

- Player position and viewing angle depend upon angle between camera and player.
- Specify angle viewing player from as fixed value.
- But then camera moves around a lot (background moves) can be disconcerting.

Dynamics: Design

- Instead, calculate angle relative to camera location (black lines).
- Only move camera if angle greater than constraints (blue lines).
- Camera will move less.
Dynamics : Design

Control size of player on screen, by controlling distance from camera to player.

5 metres

Similar to angle, often don’t want as fixed value (see next slide).

Dynamics : Design

Allow designer to set range of valid distances for camera.

Camera never gets too far from, or too close to, player.

Dynamics : Implementation

SUMMARY: Let designer control:
position of player on screen
angle looking (orientation of camera)
size (distance from camera)

Dynamics : Implementation

Calculate Angle from Camera to Target
Constrain Angle from Camera to Target
Calculate Angle from Camera to World
Constrain Angle from Camera to World
Calculate Distance from Camera to Target Plane
Constrain Distance from Camera to Target Plane

Distance to Target Plane

Target Position

Angle to Target

Angle to World
Outline

- Zoning (done)
- Dynamics (done)
- Blending (next)
- Rails

Blending: Overview

- Blending – smooth out transitions between cameras
- Three aspects:
  - Timers – track and update each blend
  - Ease – controls the smoothness of blend
  - Blend Space – defines what a blend between two cameras does

Timers: Design

- Don’t actually blend pixels
- Rather, create third camera from varying proportions of other two cameras
- Moves from first camera to second
- Position and orientation determined by blend of two cameras
- Driven by timer (started when new camera activated)
Timers: Implementation

- Timer List
  - Entry is camera fading in
  - New timers inserted at top
  - Camera can have multiple timers in list
    - This happens if player moves quickly between cameras
  - First-In, First-Out (FIFO)
  - When timer completes, all timers below it are removed

Timers: Implementation

In this example, player moves from A, to C, to B and back to C.

Zone A: list empty, start camera A
Zone C: starts new timer, camera is blend of A and C
Zone B: start new timer, camera is blend of A, B and C
Camera C's timer done, so drop camera A
Back to Zone B: start new C timer top (C in list twice)
Camera B's timer completes, drop C timer below
Camera C's timer completes, drop B below
At C 100%

Ease: Design

- Using as-is, get simple linear blend
  (see sharp corners in picture)
- When used to blend cameras, see jerk when starts to move and stops
  Can be ugly
- Want what animators call "ease"
  Feed linear blend into spline
Ease : Implementation

- **Hermite Spline**
  - Used to smoothly interpolate between key-points (e.g., camera A to camera B)
  - Fixed endpoints at P1 & P2
  - Controllable tangents
  - \( \text{ease} = [0, 1] \)
    - 0 means no ease (linear)
    - 1 means full ease
  - **Ease-in** from P1 tangent, and **Ease-out** from P2 tangent

Ease : Implementation

- B 3/4
- C 3/3
- 3/4
- **Ease** (3/4, ease, C.easeIn, B.easeOut)

Blend Space : Design

- If blend positions along straight line, will get "zoom" effect.
- Instead, blend along arc, fixed distance from player.

Outline

- Zoning (done)
- Dynamics (done)
- Blending (done)
- Rails (next)
Rails : Objectives

Want camera on a track → idea borrowed from film industry. Construct rails, put camera on little cart (a “Dolly”).

Rails : Design

• Rail can be curve (e.g., spline — numeric function compose of polynomials)
• Dolly is point on spline

Rails : Implementation

• Player is 2 units outside, so weight at p0 is 2.
• Move dolly to p1, weight is 0 since inside constraints.
• Between p1 and p2, weight stays 0.
• At p3, again 2 units outside, weight is 2.

Using weighting function, find nearest minima to previous position of Dolly.
Rails: Implementation

- Guess which direction player moved
- Take step in that direction
- If weight at new position is lower, then try another step.
- If weight is higher, turn around and go back ½ as much
- If below certain threshold, then stop.

In general, may be difficult to find minima
→ Classic hill climbing

Rails: Implementation

- Can experiment with weights
  - Distance from Player to Dolly
  - Classic drag/push camera down corridor
  - Amount Boss obscures Player
  - Number of minor characters out of frame
  - ...

- Also, can combine Dolly technique with earlier ones

Other Stuff (not Discussed)

- Dealing with multiple targets
  - Framing fights, using multiple targets
- Dynamic target definition, and calculation
  - Target changes, fade to different targets
- Overriding cameras at arbitrary points to focus on dynamic areas of interest
  - Different camera “states”
- Physical post effects like shake and sway

God of War 2 trailer: https://www.youtube.com/watch?v=GyVbK-w9pM

What techniques can you identify?