In-Game Special Effects and Lighting

Tomas Arce and Matthias Wloka
Overview – Motion Blur

- Motivation
- Our technique
- Variations
- Pros and cons
- Interactive demo
Motion Blur: Motivation

- Motion blur is a finite shutter-speed artifact

- Still photography
  - Motion blur expresses dynamic motion
  - But leaving the shutter open looks blurred, not motion-blurred (!)
  - Photographers use rear-curtain-sync flash instead
Motion Blur: Without Flash
Motion Blur: Motivation

- **Movies**
  - Non-blurred objects look jerky and stilted (see RKO Pictures’ original “King Kong” from 1933)
  - Slow-motion action sequences: pick your favorite action/adventure movie

- **Real-life**
  - Airplane propellers
  - Ceiling fans
Motion Blur: Uses in Games

- Any fighting game
  - Sword
  - Hand-to-hand
- Any game with in-game replay (think “Matrix”-style instant replay)
  - Driving games
  - Fighting games
  - Sports titles
Motion Blur: Screenshot
Motion Blur: Technique

Overview

- First render object normally
  - We will get back to why this may be necessary later

- Render object again using DirectX 8 vertex-shader
  - Stretch object from previous frame’s position to current frame’s position
  - Apply alpha-blending (similar to rear-curtain-sync flash)
Motion Blur: The Technique

- Previous Frame
- Current Frame
- Per-Vertex Motion Vectors
  - subtract
  - dual-transform
  - alpha-blend
Motion Blur: The Technique

- Take vertex to view-space
  - Using current frame’s transform
  - As well as the previous frame’s transform
- The difference is a motion vector M
- If \((M \cdot N > 0)\) then vertex faces into the motion
  - Transform it using the current frame’s transform
- Else vertex faces away from motion
  - Use the previous frame’s transform
  - Force the vertex into semi-transparency
Motion Blur: Some Gotcha’s

- Use of transparency requires
  - Back-to-front for non-convex objects
    - Create six index lists sorted corresponding to view-direction aligned w/ object’s ±x, ±y, ±z axis
    - Choose index-list closest to actual view direction
  - When object moves away from camera
    - All motion-front-facing vertices, i.e., all opaque vertices, are camera-back-face culled
    - Object seems to disappear
    - Thus, first render object normally
Motion Blur: Necessity of First Pass

- Previous Frame
- Current Frame

Per-Vertex Motion Vectors (pointing into the screen)

subtract

dual-transform

alpha-blend
Motion Blur: Single Pass Artifact
Motion Blur: Variations

- Emphasize/De-emphasize motion blur
  - Artificially lengthen/shorten motion vector
- Transparency of motion-back-facing vertices
  - Must vary with length of motion vector: if motion vector zero, vertices must be opaque
    - $\text{MAX}(0.1, 1 - \text{length(motion vector)}/\text{extent})$ works well
  - Try non-linear equations
Motion Blur: Pros

- General technique
  - Applies to any object
  - Takes camera movement into account
- Two pass technique, possibly one pass
  - Runs fully on GPU for GeForce 3 and up
- Looks like motion blur
  - Not multiple (jagged) renderings
Motion Blur: Vs. Multiple Rendering
Motion Blur: Cons

- Potentially wrong visibility
- Per-vertex operation: no texture blurring
  - Lowering texture LOD-level looks fine
  - Pre-process textures in motion-directions
- Linear interpolation from frame to frame
  - Fast rotating objects problematic
- Uses transparency, i.e., back-to-front rendering
Overview – Depth of Field

- Motivation
- Our technique
- Variations
- Pros and cons
- Interactive demo
Depth of Field: Motivation

- Depth of field: aperture-based artifact
  - Virtually all still- and motion-picture photography uses aperture-based lenses
  - Depth of field is shallower
    - The wider the aperture (low light or fast shutter)
    - The longer the focal-length (telephoto)
    - The closer the focusing distance
- Depth of field focuses attention
  - Look at any movie or professional photograph
Depth of Field: Potential Uses in Interactive Games

- Anytime there is a naturally limited focus
  - Surrounding environment is non-essential
  - Have the environment come in- and out-of-focus as players change position or camera parameters adjust
- Careful: frustration if player ever has to discern anything out-of-focus
- In-game movie sequences
- In-game replay (mimicking TV-coverage)
Depth of Field: Screenshots
Depth of Field: Technique Overview

- Render scene to texture
  - Vertex shaders compute distance to camera
  - Pixel shader uses interpolated camera-distance to look up “blurriness interpolator”
  - Stores that interpolator in texture’s alpha
  - Copy and blur the texture multiple times
- Blit texture-target to back-buffer
  - Pixel shader chooses between original and blurred versions based on blurriness interpolator
Depth of Field: Render Scene to Texture

- Match texture dimensions to back-buffer
  - Non-power of two texture
  - Sub-rect of next larger power-of-two texture

- Every object uses a vertex shader to compute distance to camera
  - Radial distance is correct, takes 3 instructions
  - Linear z-distance looks similar (especially for tele-photo lenses), takes 1 instruction
  - Normalize to $[0, 1]$ and output as texture coordinate
Depth of Field: Render Scene

- Every object uses a pixel shader to
  - Transform interpolated camera-distance to blurriness interpolator
  - Store blurriness interpolator in texture-alpha
Depth of Field: Circle of Confusion

- World-points map to dots on film-plane
- Diameter of dot: circle-of-confusion
  - Measure of blurriness
- Circles of diameter < $\varepsilon$ are “in-focus”
- Formula
  
  \[
  C(d, f_D, f_L, f_S) = \text{abs}\left(\frac{f_D}{d} - 1\right) \cdot \frac{f_L}{f_S}(f_D - f_L)
  \]
  
  $d$: distance to camera, $f_D$: camera’s focus distance
  $f_L$: camera’s focal length, $f_S$: camera’s f-stop
Depth of Field: Blurriness

- Original texture render-target corresponds to circles of confusion of $\varepsilon$ or less
- Maximally blurred texture render-target corresponds to circles of confusion of $E$ or more
- Generate texture $T$
  \[ T.r(d) = \frac{C(d, fD, fL, fS) - \varepsilon}{(E - \varepsilon)} \]
Depth of Field: Copy and Blur Textures

- Map texture onto screen-covering quad
- Render the quad to texture
  - Lower the resolution
    - Copying and blurring becomes faster
    - But introduces unique artifacts
  - Filter-blit
    - Choice of filters: box, cone, etc
    - Roughly as fast as rendering texture w/o filtering
- Repeat
Depth of Field: Filter Blit (1/4)

- Image sampling:

- All pixels are sampled the same
Depth of Field: Filter Blit (2/4)

- All stages use the same texture
- Quad’s UV-coordinates provide offset

Neighbor to the top
Neighbor to the right
Neighbor to the bottom
Neighbor to the left
Depth of Field: Filter Blit (3/4)

- Four texture units allow 4 samples:
  ![Texture Samples](image)
  
  - Bilinear texture-sampling does right things
  - Filter kernel may be arbitrarily rotated
  - Can assign arbitrary weights to samples:
    \[ r_0 = c_0 t_0 + c_1 t_1 + c_2 t_2 + c_3 t_3 \]
  - Blur, sharpen, diagonal edge-detection, etc.
Depth of Field: Filter Blit (4/4)

- Take advantage of bilinear texture-interpolation
- Weights are assigned implicitly

\[ T_0 = \frac{a+b+d+e}{4} \]
\[ T_1 = \frac{b+c+e+f}{4} \]
\[ T_2 = \frac{d+e+g+h}{4} \]
\[ T_3 = \frac{e+f+h+i}{4} \]

\[ R_0 = \frac{T_0+T_1+T_2+T_3}{4} = \frac{e}{4} + \frac{b+d+f+h}{8} + \frac{a+c+g+i}{16} \]
Depth of Field: Render to Backbuffer

- Texture 0, contains
  - RGB: scene
  - Alpha: blurriness interpolator
- Texture n, is texture 0 blurred n-times
- Texture 2n, is texture n blurred n-times
- Render screen-size quad to back-buffer
  - Tex 0 = texture 0
  - Tex 1 = texture n
  - Tex 2 = texture 2n
Depth of Field: Variations

- Use 1D, 2D, or volume texture for the blurriness interpolator look-up
  - $T.r(d) = C(d, fD, fL, fS) - \varepsilon) / (E - \varepsilon)$
  - $T.r(d, fD) = C(d, fD, fL, fS) - \varepsilon) / (E - \varepsilon)$
  - $T.r(d, fD, fL) = C(d, fD, fL, fS) - \varepsilon) / (E - \varepsilon)$

- Texture blurring
  - Filter type, e.g., 9-sample box
  - Variable n, e.g., 5
  - Resolution reduction, e.g., none

- Blurriness interpolator interpolation
Depth of Field: Pros

- General technique
  - Takes camera’s f-Stop, focal-length and distance into account
  - Allows different parts of a model (triangle even) to be in- or out-of-focus
  - Blurs textures and geometry
  - No pre-processing required
- Scene needs to be rendered only once
  - Runs fully on GPU
Depth of Field: Cons

- Not physically accurate
  - Visibility on blurred edges potentially wrong

- Limitations
  - Maximum blur depends on how much original texture is processed
  - Camera-distance only computed per-vertex
  - Alpha-channel precision (8bits) for camera-distance
  - Final pass only uses 3 blurriness textures

- No fixed-function rendering
Further Reading

Further Reading

- David M. Jacobsen, “Photographic Lenses Tutorial,”
Questions...

http://developer.nvidia.com/
Extra Slides
Motion Blur: Prior Art in Games

- Any sword-fighting game...
  - Zelda
  - Soul Calibur
  - ...
- Jet Grind Radio Future
- Blur is pre-modeled and calculated
  - Not a general technique
  - Too expensive to use on everything
; Transform position into view-space with previous worldview-transform
dp4 r0.x, v0, c[CV_PREV_WORLDVIEW_EW_TXF_0]
dp4 r0.y, v0, c[CV_PREV_WORLDVIEW_EW_TXF_1]
dp4 r0.z, v0, c[CV_PREV_WORLDVIEW_EW_TXF_2]

; Transform position into view-space with current worldview-transform
dp4 r1.x, v0, c[CV_CURR_WORLDVIEW_EW_TXF_0]
dp4 r1.y, v0, c[CV_CURR_WORLDVIEW_EW_TXF_1]
dp4 r1.z, v0, c[CV_CURR_WORLDVIEW_EW_TXF_2]

; the transform difference in view-space is the motion vector
Motion Blur: The Vertex Shader (2/4)

; artificially shorten (lengthen) this motion vector
mul r2.xyz, r2, BLUR_FRACTION

; transform normal into view-space
dp3 r3.x, v3, c[CV_CURR_WORLDVIEW_IT_0]
dp3 r3.y, v3, c[CV_CURR_WORLDVIEW_IT_1]
dp3 r3.z, v3, c[CV_CURR_WORLDVIEW_IT_2]

; dot the motion vector with the projected vertex normal
dp3 r2.w, r2, r3
Motion Blur: The Vertex Shader (3/4)

; the result of the dot-product decides which transform we use
slt r3.w, r2.w, ZERO
mad r4.xyz, r3.w, -r2, r1
expp r4.w, v0.x ; generate constant 1.0

; compute final position by transforming r4 to clip-space
dp4 oPos.x, r4, c[CV_PROJ_TXF_0]
dp4 oPos.y, r4, c[CV_PROJ_TXF_1]
dp4 oPos.z, r4, c[CV_PROJ_TXF_2]
dp4 oPos.w, r4, c[CV_PROJ_TXF_3]
Motion Blur: The Vertex Shader (4/4)

; compute alpha component depending on length of motion
; vector
dp3 r2.w, r2, r2
rsq r1.w, r2.w
mul r2.w, r2.w, r1.w ; r2.w now contains length(motion vec)
; now compute r2.w = 1 - length(motion vec)/extent
mad r2.w, -r2.w, c[CV_OBJECT_EXTENT].x,
    c[CV_OBJECT_EXTENT].y
; clamp color and alpha to minimum values
max r5, c[CV_PREV_COLOR], r2.w
Depth of Field: Prior Art in Games

- NFL 2K for Sega Dreamcast
  - Focusing on players after plays

- Lowers texture LOD-bias for all far-away objects
  - Works great only for screen-aligned sprites
  - Cannot focus on part of an object
Depth of Field: Vertex Shader

; compute z-linear distance (instead of radial distance)
dp4 r0.z, v0, c[CV_WORLDVIEW_EW_2]

; subtract mMinDistance & divide by maxDistance-minDistance
; c[CV_MINMAX_DIST].x = mMinDistance / (mMaxDistance-mMinDistance)
; c[CV_MINMAX_DIST].y = 1.0f/(mMaxDistance-mMinDistance)
mad oT0.x, r0.z, c[CV_MINMAX_DIST].y, -c[CV_MINMAX_DIST].x

; copy current focus distance & focal length to texture coord
mov oT0.yz, c[CV_FOCUS_CONST].xxyy
Depth of Field: Final Pixel Shader

def c0, 0.0f, 0.0f, 0.0f, 0.5f
text t0
text t1
text t2

; interpolate interpolator: straight t0 produces ghosting
; (DoF selection is hi-res (ie, t0) even for blurred parts).
lrp r0.a, c0, t2.a, t0.a
mov_x2_sat r1.a, r0.a // pretend 0 <= r0.a <= 0.5
lvp r1.rgb, r1.a, t1, t0 // interpolate t0, t1 &
store
mov sat r1.a, r0.bx2.a // pretend 0.5 <= r0.a
Depth of Field: Interactive Demo
Putting It Together

Demo courtesy of

INEVITABLE™

www.inevitable.com

Thanks to Tomas Arce