Frame Buffer Postprocessing Effects in DOUBLE-S.T.E.A.L (Wreckless)

Masaki Kawase

BUNKASHA GAMES

BUNKASHA PUBLISHING CO.,LTD

http://www.bunkasha-games.com
http://www.daionet.gr.jp/~masa/

Today's Contents

- Xbox DirectX
- Fake HDR and Glare filters
- Depth of Field (DOF)
- Post-processing image filters

Xbox DirectX

- Xbox DirectX Extensions
  - Same as GeForce3 OpenGL Extensions
    - Texture shader
    - Register combiners
    - Shadow mapping
- Capability to typecast resources
  - Use D3DFMT_D2S8 depth-buffer as a 3DFMT_A8R8G8B8 texture
  - Render to a Vertex Buffer

Pixel Shader Extensions

- Register combiners for Pixel Shader
  - General combiners
    - Color blending instructions
  - Final combiner
    - Fog blending
    - Specular add

General Combiners (1)

- xmma d0,d1,d2, s0,s1,s2,s3
  - d0 = s0*s1
  - d1 = s2*s3
  - d2 = s0*s1 + s2*s3

- xmmc d0,d1,d2, s0,s1,s2,s3
  - d0 = s0*s1
  - d1 = s2*s3
  - d2 = (r0.a>0.5) ? s2*s3 : s0*s1

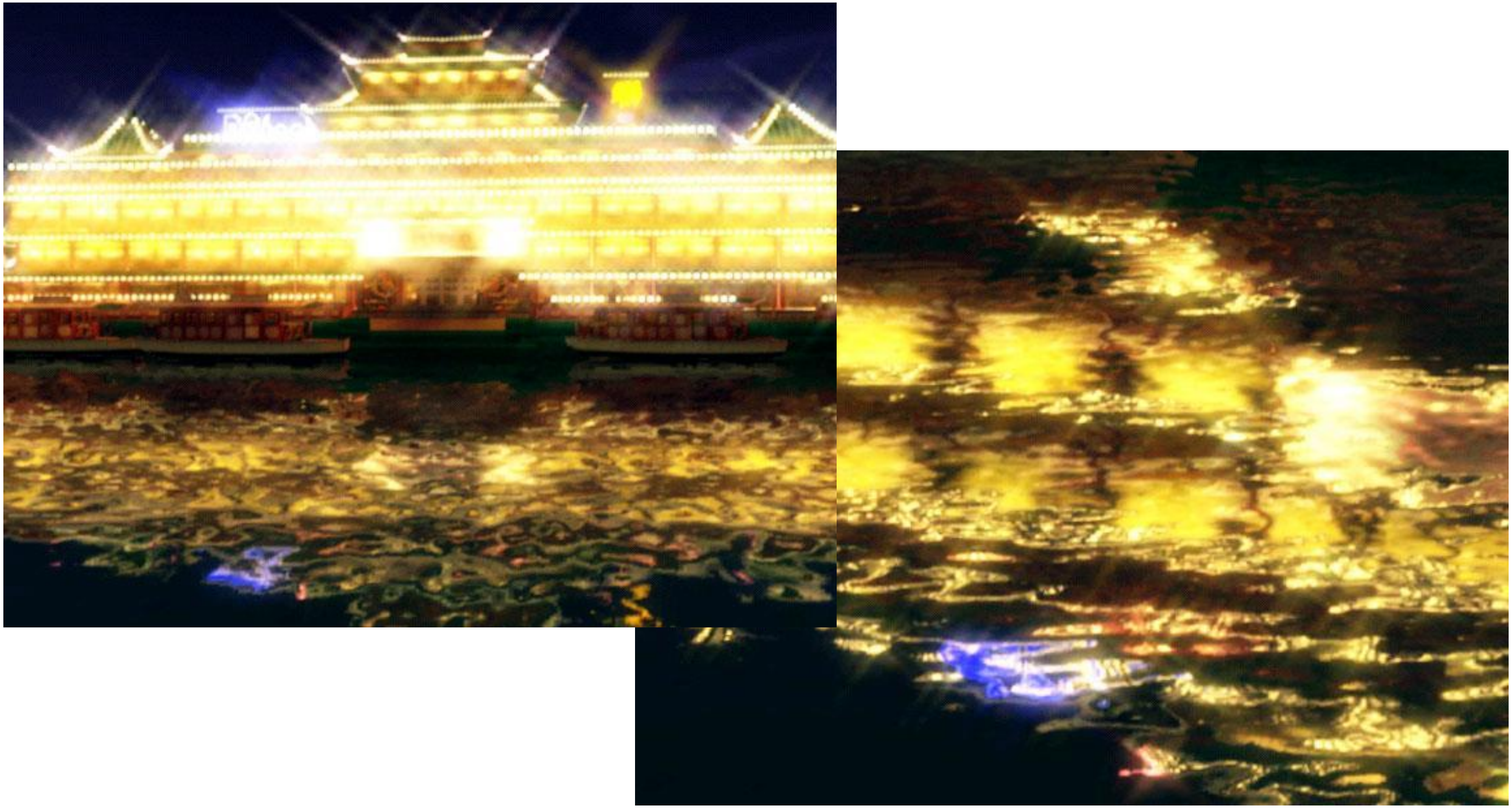
General Combiners (2)

- xdd d0,d1, s0,s1,s2,s3
  - d0 = s0 dp3 s1
  - d1 = s2 dp3 s3
- xdm d0,d1, s0,s1,s2,s3
  - d0 = s0 dp3 s1
  - d1 = s2*s3

Final Combiner

- xfc s0,s1,s2,s3, s4,s5, s6
  - Final output rgb = s0*s1 + (1-s0)*s2 + s3
  - Final output alpha = s6
  - Final combiner special input registers
    - PROD = s4*s5
    - SUM = r0+v1
    - FOG.a = fog factor

Fake HDR and Glare filters



High Dynamic Range (HDR) Rendering

- Very important in representing real-world brightness
- Very bright scene causes “Glare”

HDR Rendering Process

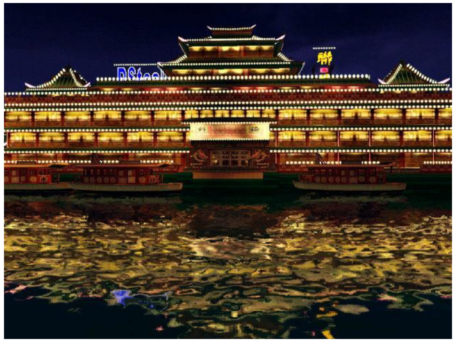
- Render scene with HDR
- Generate glare images from bright pixels
- Add glare to Frame Buffer

HDR Scene Rendering with A8R8G8B8 Frame Buffer

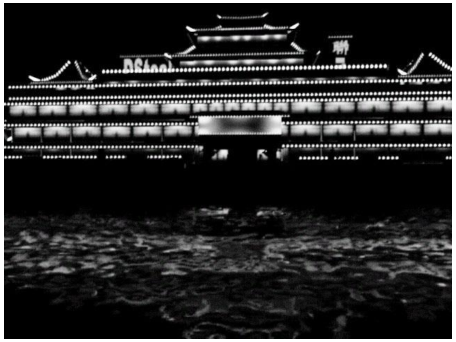
- Glare effects need HDR
- Use alpha channel as an additional information about pixel brightness
  - Render scene with alpha channel
  - Output higher alpha values to bright pixels

Glare-generation Process

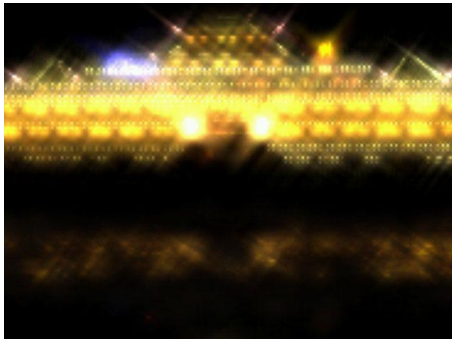
Frame buffer



Frame buffer alpha

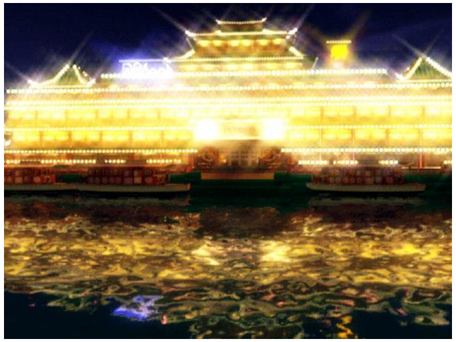


+



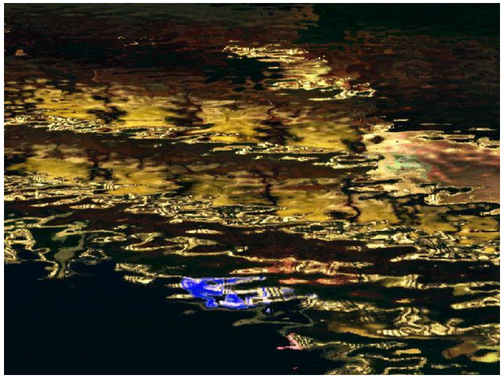
Generate glare

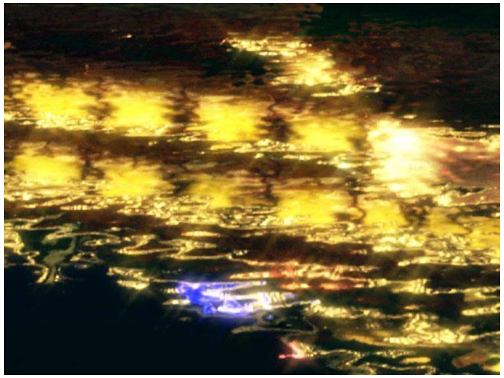
=



Final image

Frame buffer





Final image

Glare filters

- Downsample frame buffer to ¼ * ¼ (1/16) the size
- Pixel brightness = RGB * A
- Generate glare
  - Afterimage
  - Bloom
  - Star (light streaks)
  - Ghost (not used in DOUBLE-S.T.E.A.L)

Afterimage

- Update afterimage

  Next afterimage =

  Previous frame afterimage * p +
  current frame image * c – 1/255
  p: previous image weight
  c: current image weight

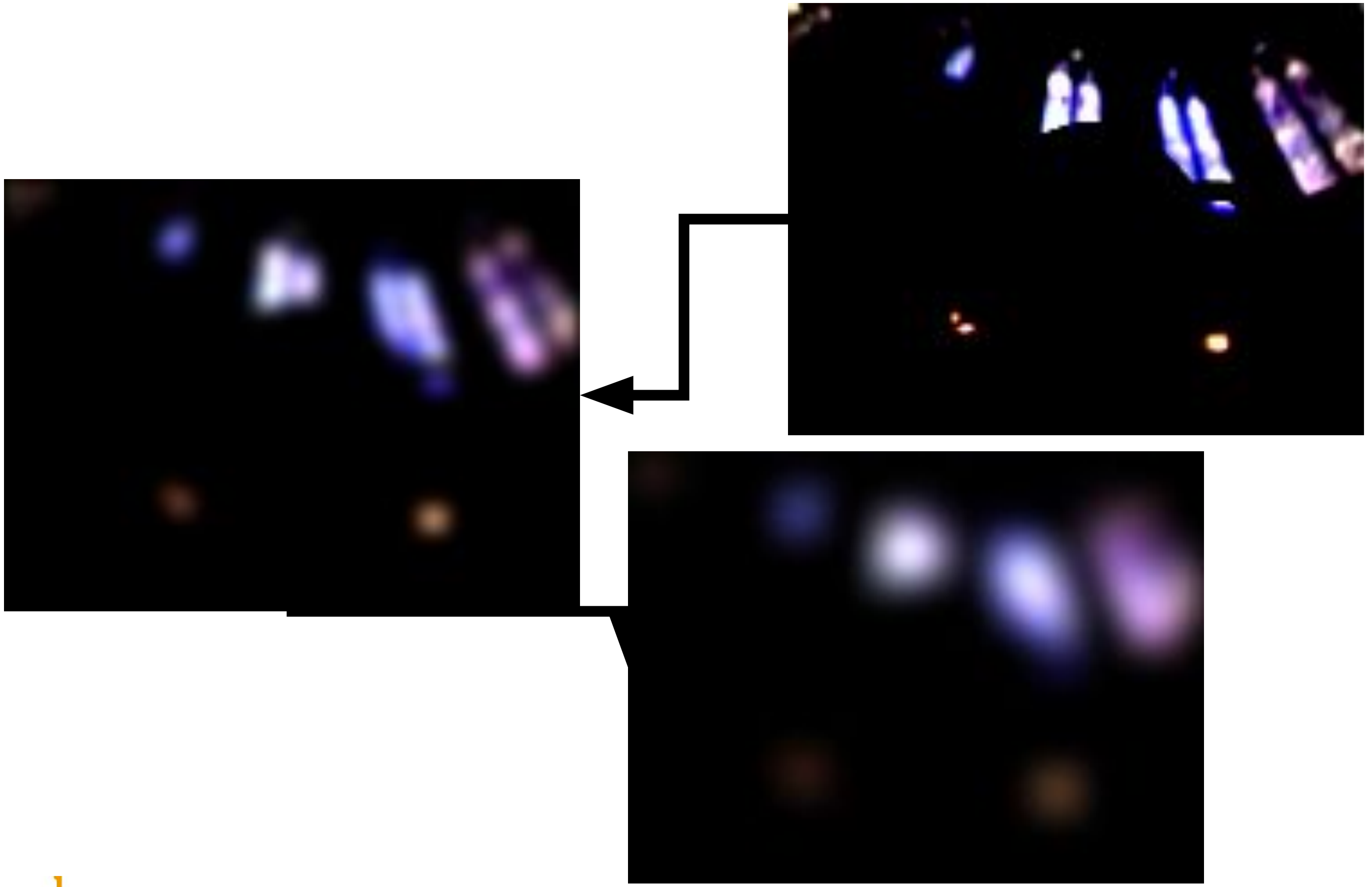
- It's not LERP (Linear intERPolation)
  - p+c can be greater than 1.0
    - e.g.

      p = 0.9
      c = 0.25

- Bias –1/255
  - Prevent dirty pixels from remaining

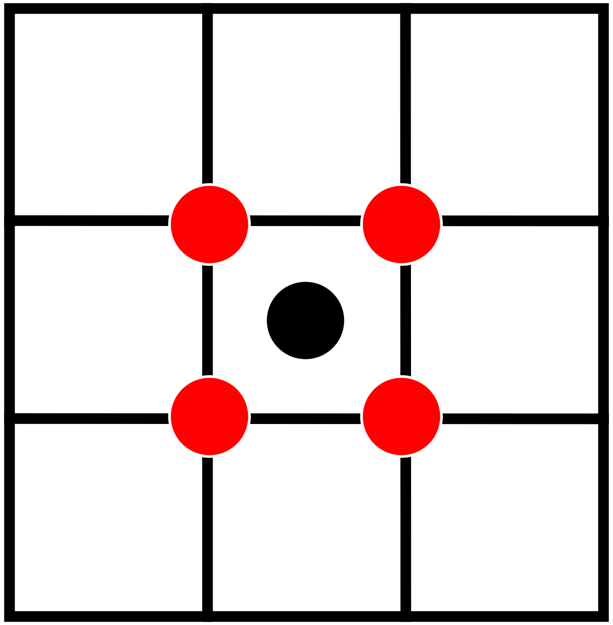
Bloom

- **Repeatedly apply small blur filters**



Bloom filter (1^{st} pass)

1^{st} pass

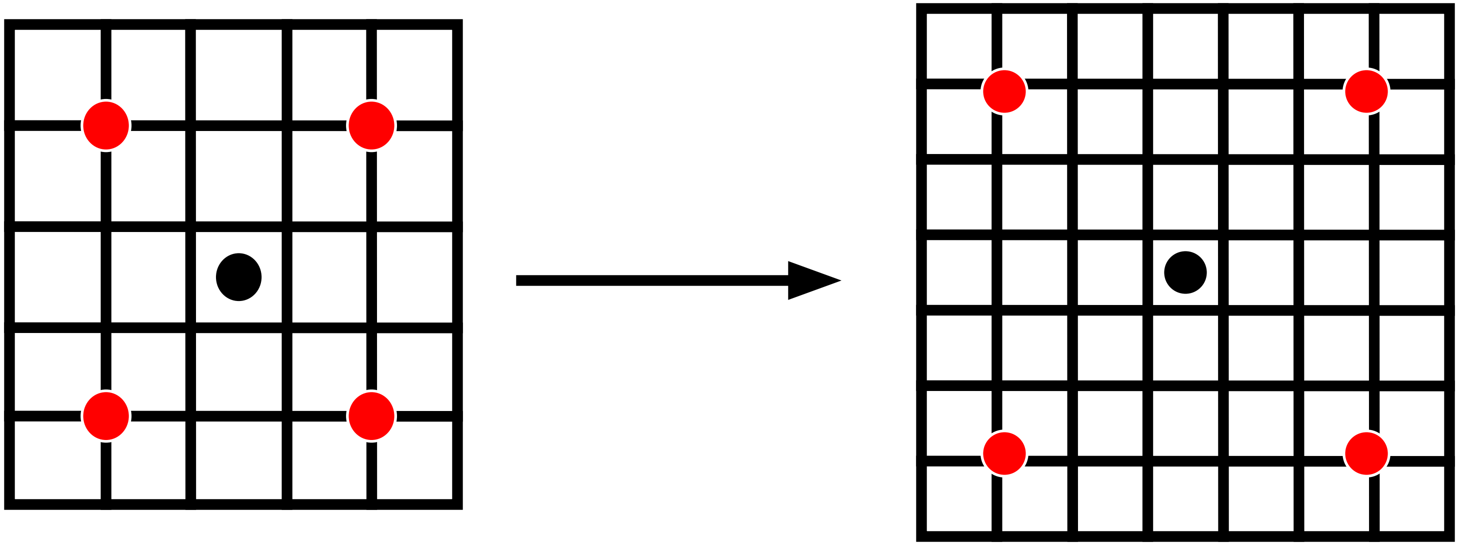


1/16	2/16	1/16
2/16	4/16	2/16
1/16	2/16	1/16

● Pixel being Rendered

🔴 Texture sampling points

Bloom filter (2^{nd}, 3^{rd}, … pass)



2^{nd} pass

- SetTexture(0-3, 1st render target) ;

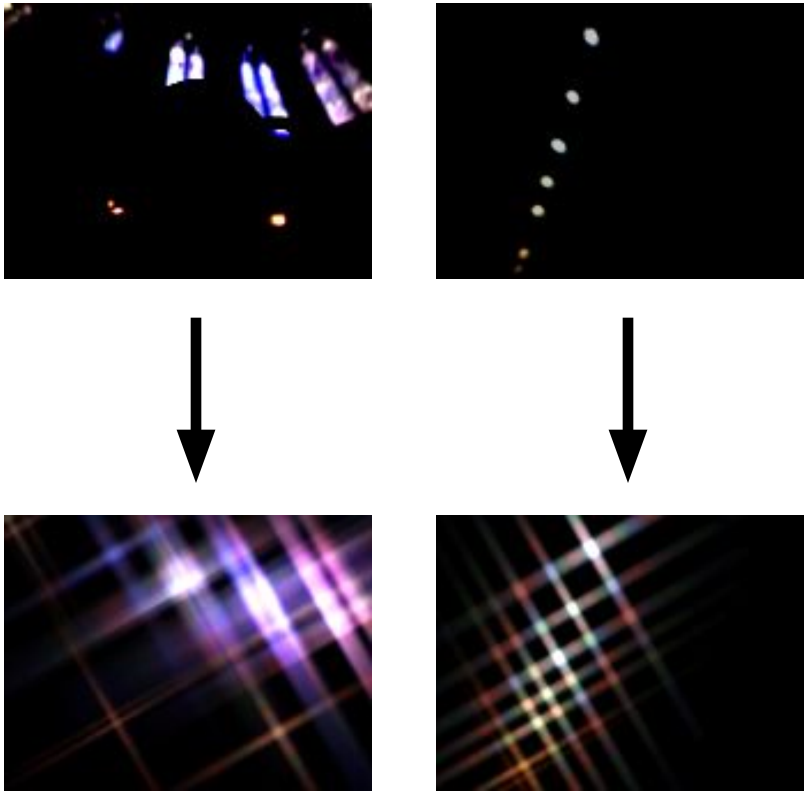
3^{rd} pass

- SetTexture(0-3, 2nd render target) ;

Repeat as many times as you like

Star (light streaks)

- Caused by diffraction or refraction of incoming light
  - Cross filter
  - Stop (Diaphragm blades)

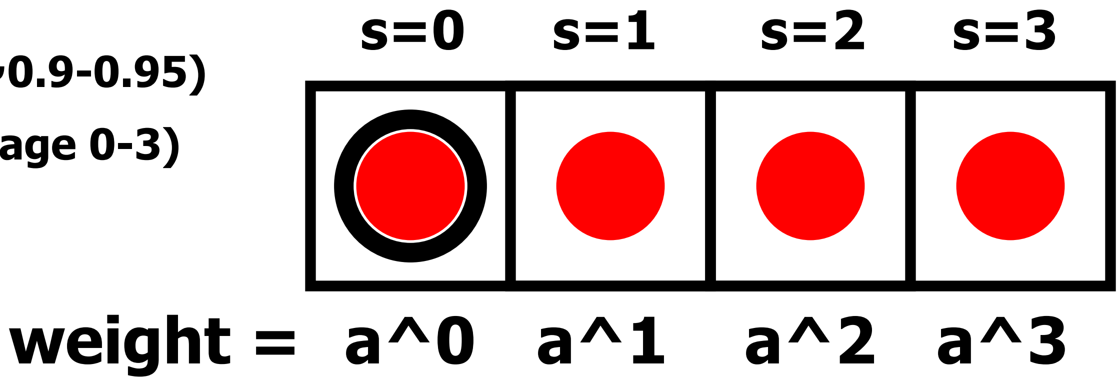


Light streak (1st pass)

- Texcoord[s] = rendering point + s texels
- color weight[s] = a^s

a: attenuation(about ~0.9-0.95)

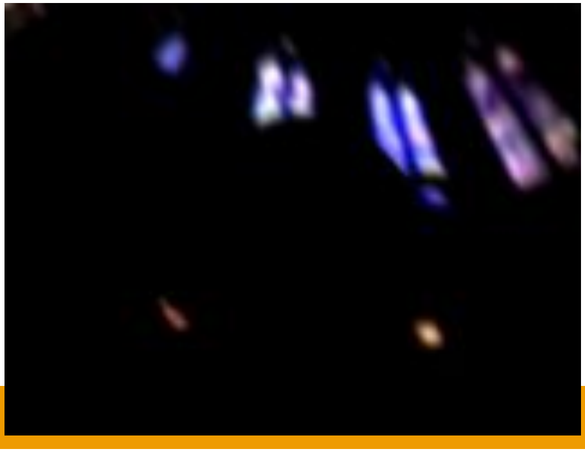
s: sampling (texture stage 0-3)



s=0 s=1 s=2 s=3

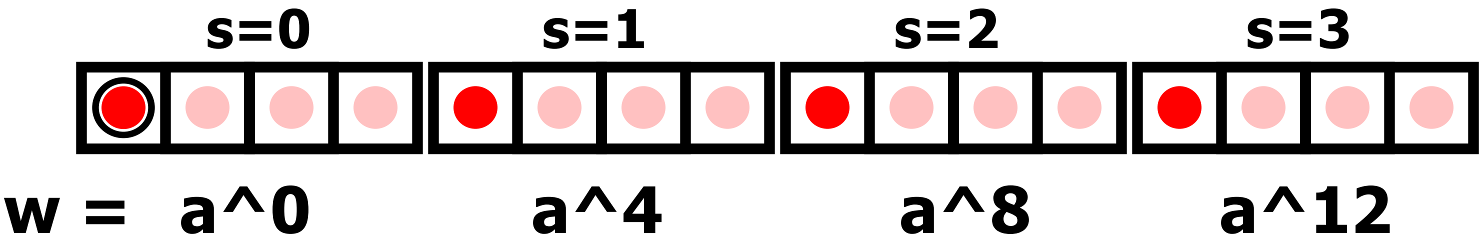
weight = a^0 a^1 a^2 a^3

4-pixel blur



Light streak (2nd pass)

- SetTexture(s, 1st render target) ;
- Texcoord[s] = rendering point + 4*s
- color weight[s] = a^(4*s)



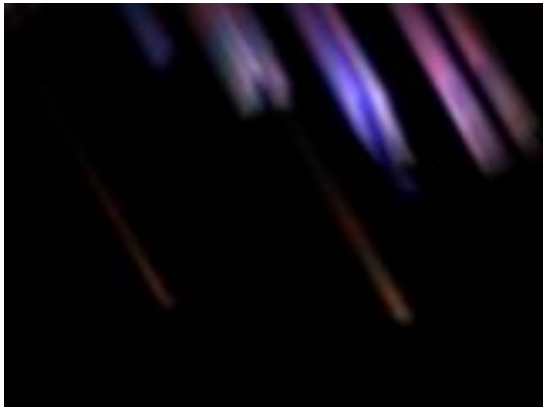
16-pixel blur

Light streak (n^{th} pass)

- n^{th} pass
  - SetTexture(s, n-1th render target) ;
  - b = 4^(n-1)
  - Texcoord[s] = rendering point + b*s
  - color weight[s] = a^(b*s)
- Modulate color for spectral dispersion

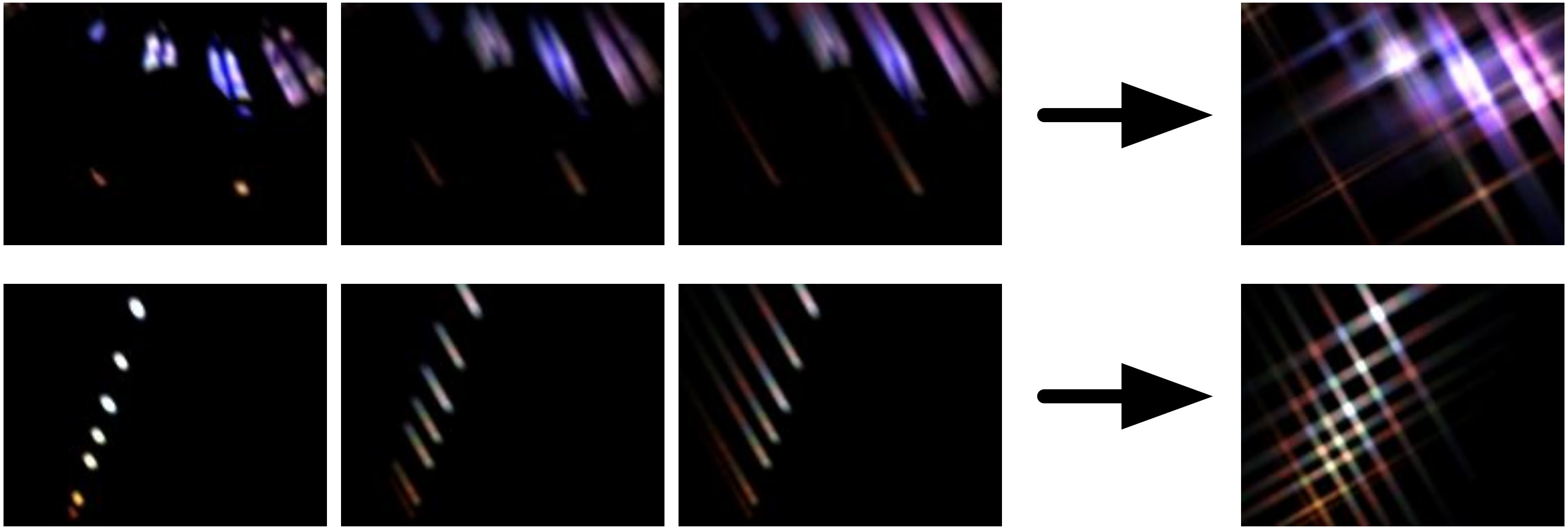
- 4^n-pixel blur
  - n=2 or 3 for good results

3rd pass
64-pixel blur



Repeat the above process

- 2, 4, 6 or 8 directions



1st pass

2nd pass

3rd pass

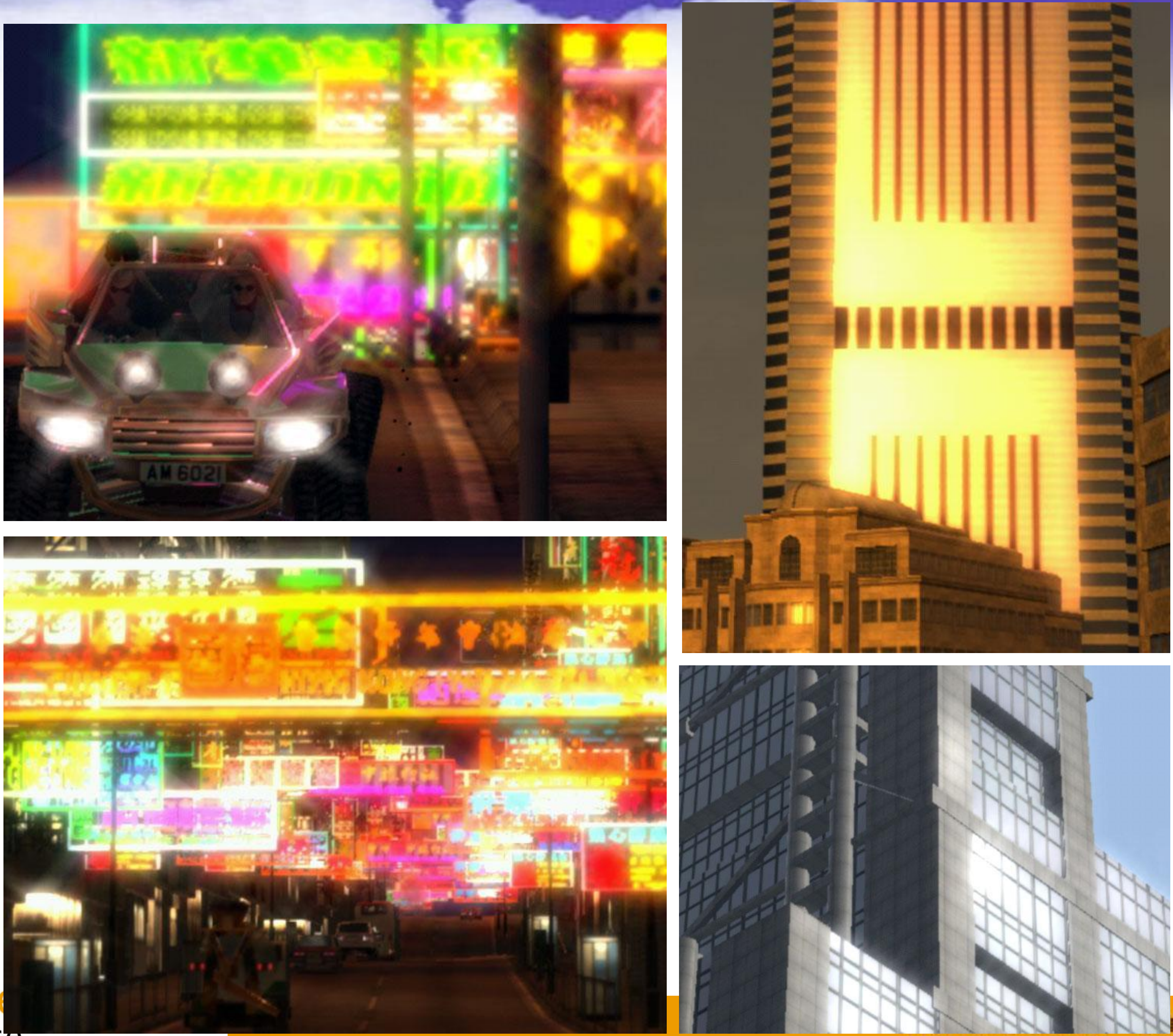
x4 directions

4 pixels

16 pixels

64 pixels

64*4 pixels


AM 6021

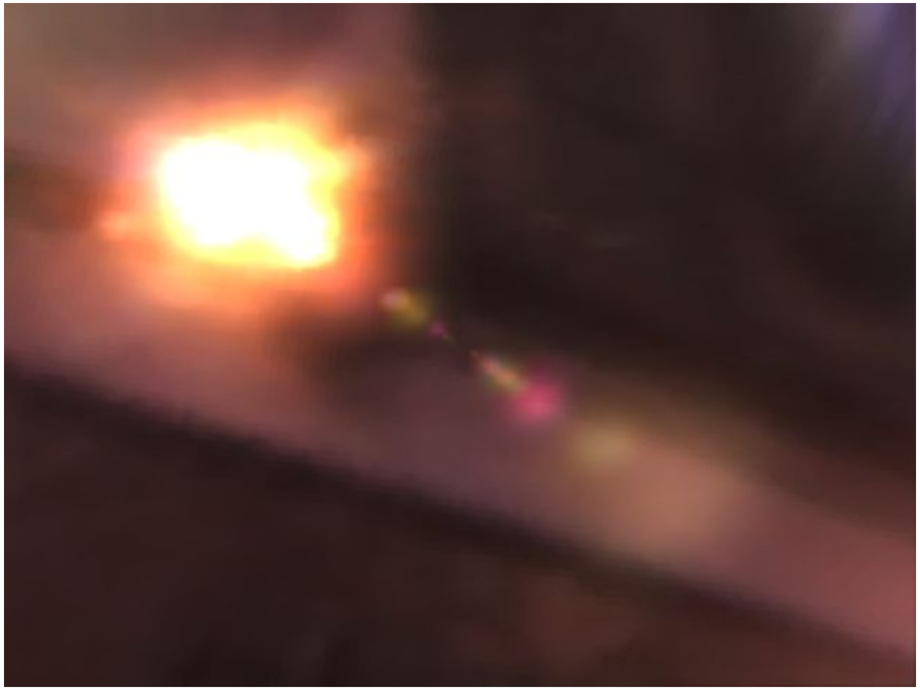
POLICE
HD 8692
REPLAY

TAXI

TIME: 01:54.16
45
23 MPH
1st

Ghost
(not used in DOUBLE-S.T.E.A.L)

- Caused by internal reflections inside the lens system
- Scaling about the screen center



Scaling about the screen center

- texcoord = (original texcoord - 0.5) * s + 0.5

  s: An arbitrary scaling factor

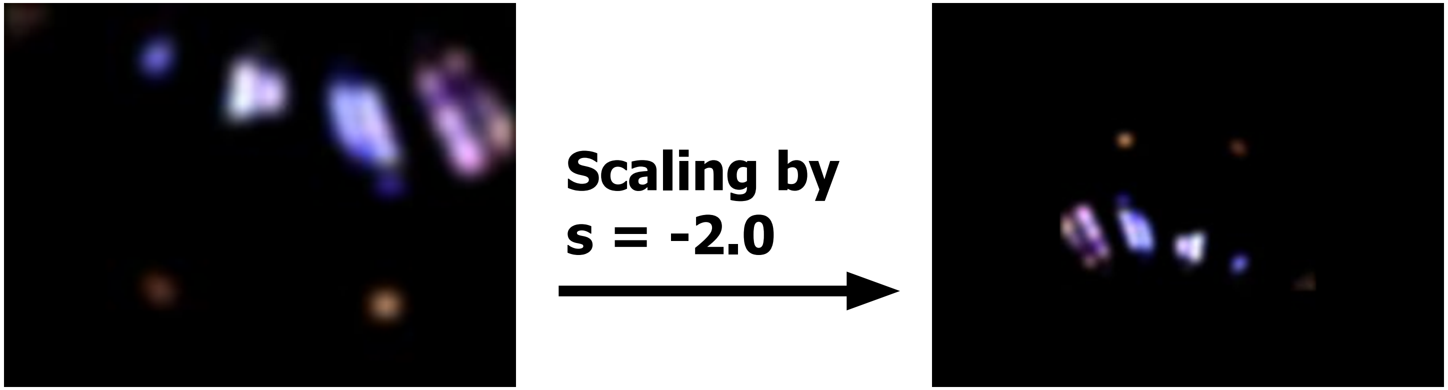
e.g. scaling by s = -2.0

original texcoords

(0,0) (1,0)

scaled texcoords

(-0.5,-0.5) (1.5,-0,5)



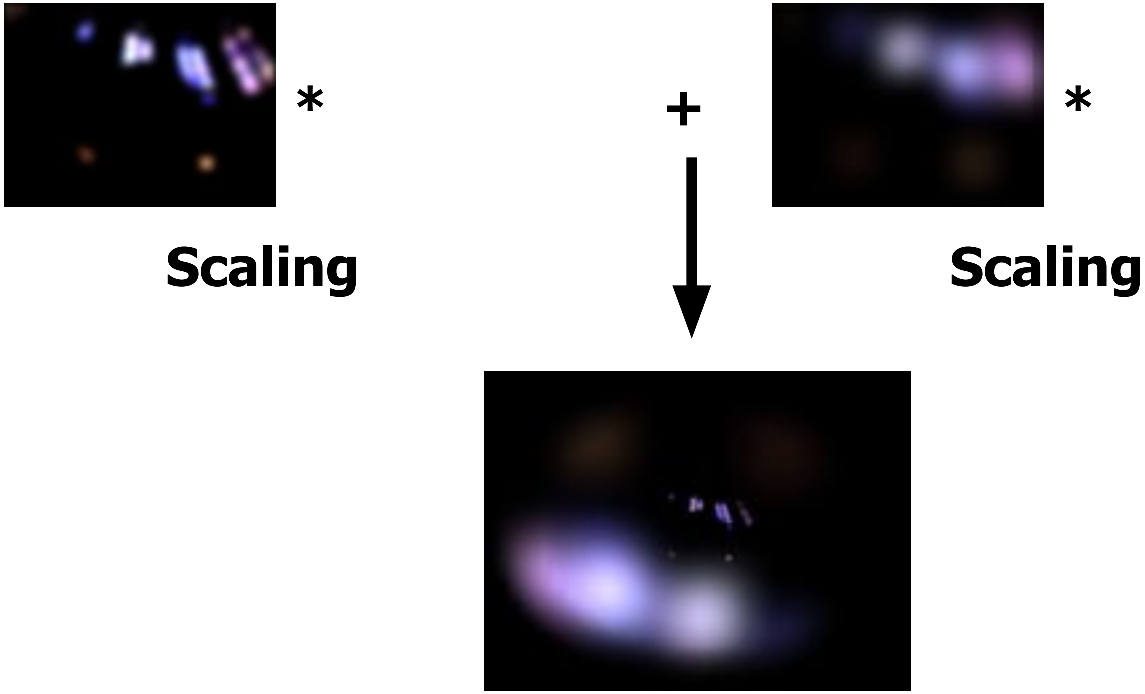
Scaling by s = -2.0

(0,1) (1,1)

(-0.5,1.5) (1.5,1.5)

Ghost (1st pass)

- Mask the source images with a smooth circle
  - To prevent rectangular edges

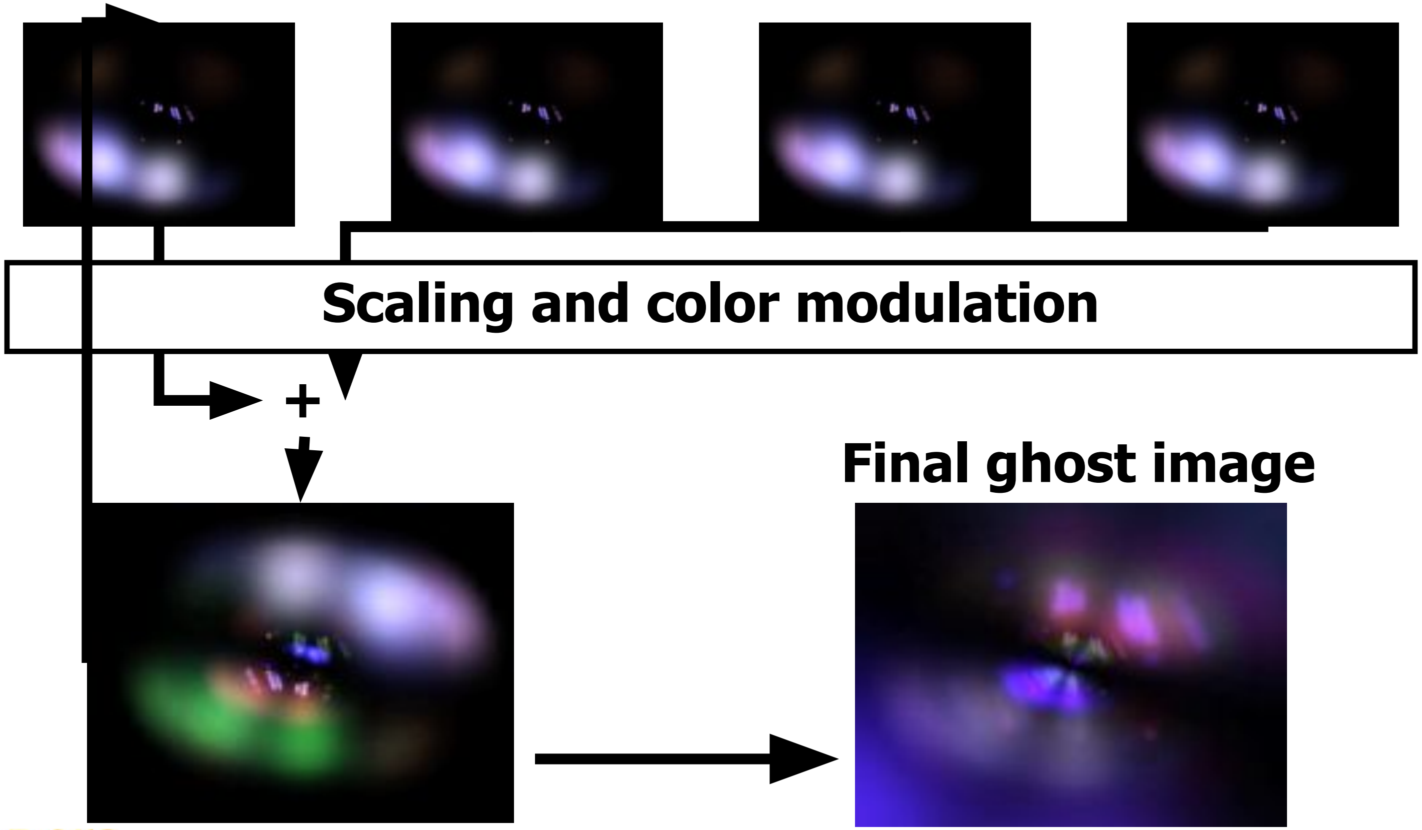


Scaling

Scaling

Ghost (2nd and 3rd passes)

- **Multi-tap scaling and color modulation**

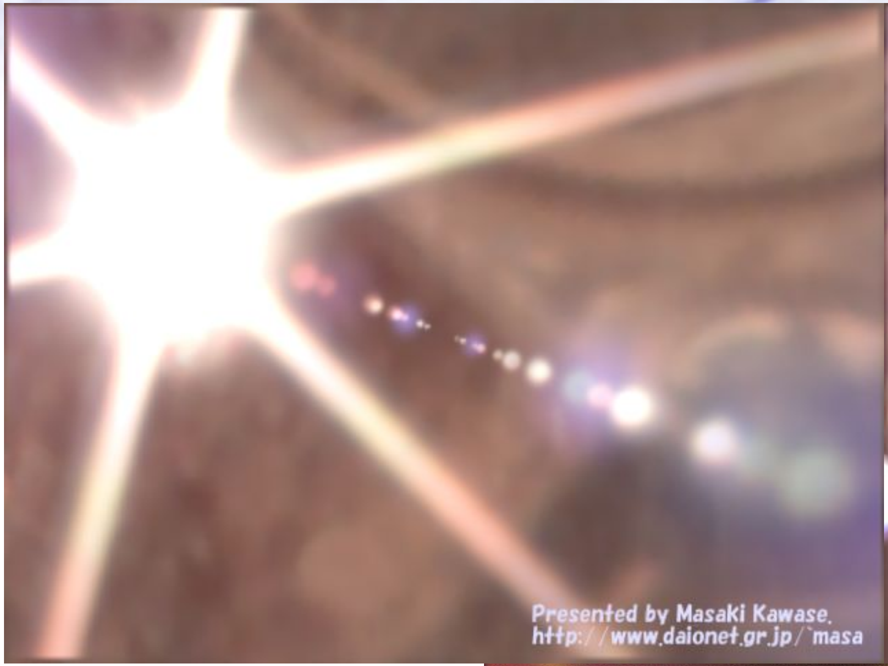


Scaling and color modulation

Final ghost image

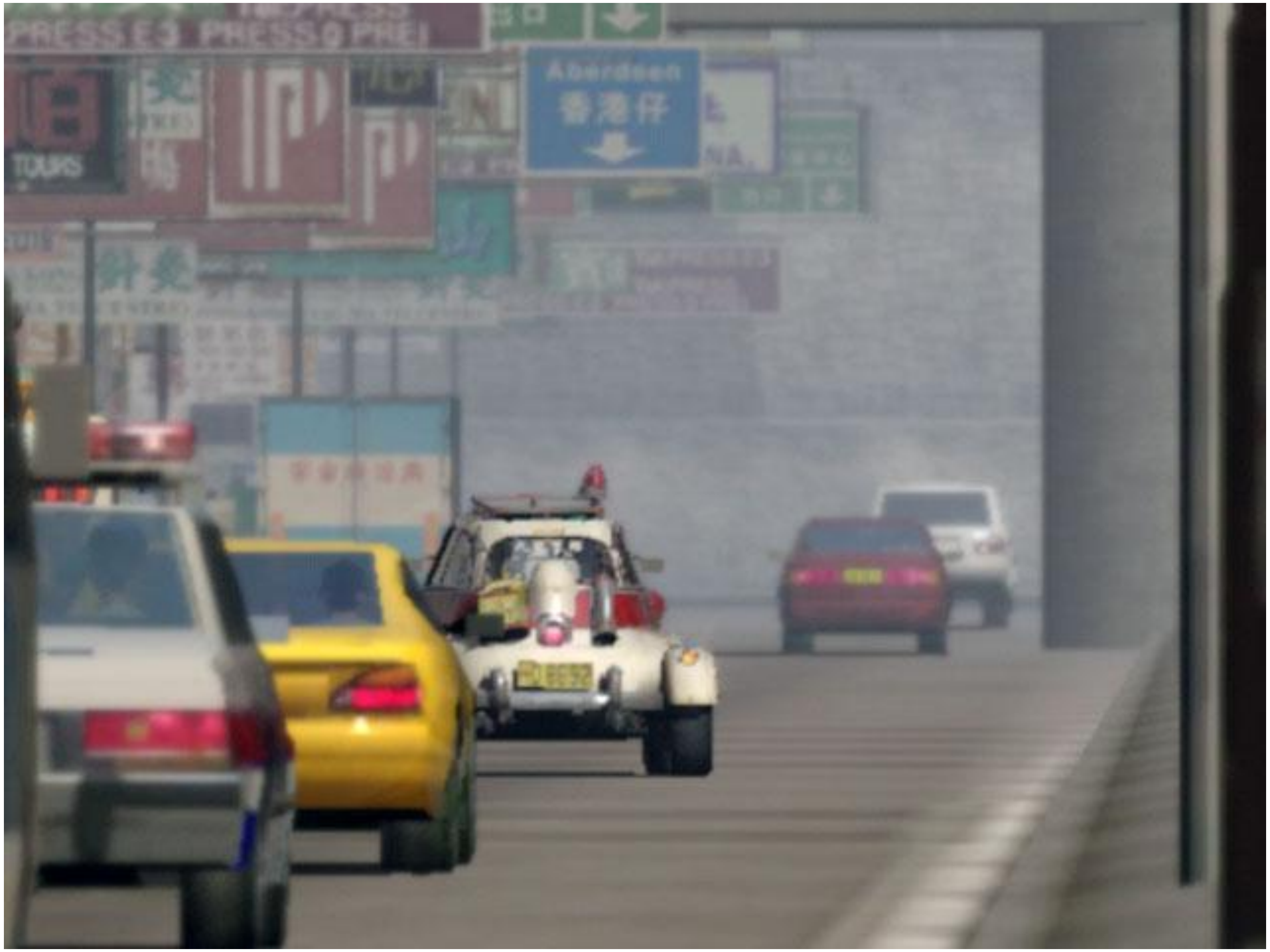
Glare Effects in DirectX9

- High-Precision buffer formats
  - True HDR frame buffer
- More complex pixel operations
  - Gorgeous glare effects
  - Still too expensive for games
    - Will hopefully be of practical use in the near future


Presented by Masaki Kawase.
http://www.daionet.gr.jp/~masa

Presented by Masaki Kawase.
http://www.daionet.gr.jp/~masa

Depth of Field (DOF)



DOF Process

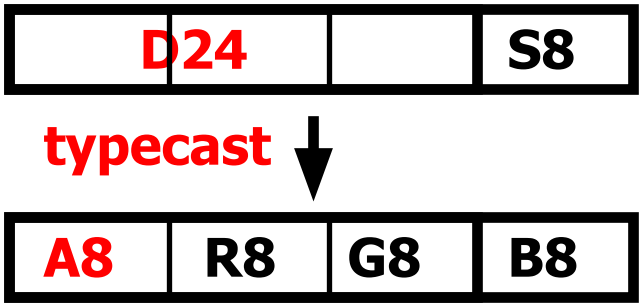
- Use depth buffer
  (W buffer in DOUBLE-S.T.E.A.L)
  - Generate blurred frame buffer
    - Can be smaller than the frame buffer
    - But don’t use box filter to resize
      - Multi-tap blur filter is recommended
  - Calculate screen-space blurriness based on W buffer and focal distance per pixel
  - Blend the blurred image and the frame buffer based on the blurriness

Look up screen-space blurriness from W buffer

- **Calculating screen-space blurriness per pixel is a bit complex operation**
  - **Directly look up blurriness from W value**
  - **1D look-up texture**
    - **8-bit table for mapping the W value to blurriness**
    - **Calculate blurriness for each W value by CPU**
      - **8 bits : 256 elements per frame**

Lookup screen-space blurriness

- **Sample D24S8 W value as A8R8G8B8 texture**
  - Can get the highest 8 bits of depth component as 8-bit alpha by typecasting
- **Use dependent texture read "texreg2ar"**
  - "texreg2ar" uses alpha and red components of another texture as the current texture coordinates (u,v)



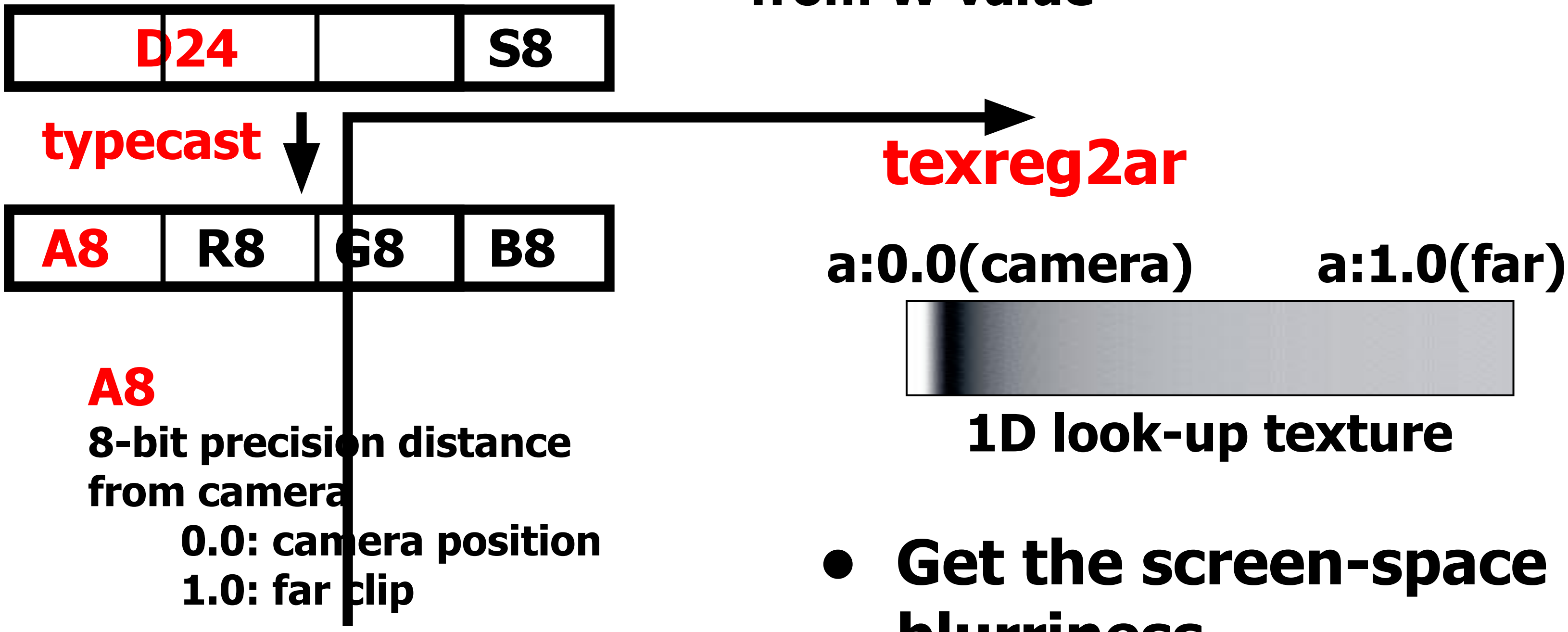
D24 | S8

typecast

A8 | R8 | G8 | B8

Lookup blurriness

Look up screen-space blurriness from W value



1D look-up texture

A8

8-bit precision distance from camera

0.0: camera position
1.0: far clip

- Get the screen-space blurriness

Blending based on blurriness

- Use two blurred images in addition to the original frame buffer
  - One is a bit blurred and the other is strongly blurred
  - The DOF result is a blend of three images (the frame buffer and two blurred images)
  - Better than a blend of two images (the frame buffer and one blurred image)
- The blurred images can be small
  - 256x192 and 160x120 in DOUBLE-S.T.E.A.L

Calculate Blend factor Alpha and Color (1)

- Calculate "Blend Alpha" and "Blend Color" in Pixel Shader
- Alpha-blend with the frame buffer
  - Blend Alpha should always be smaller than 1.0

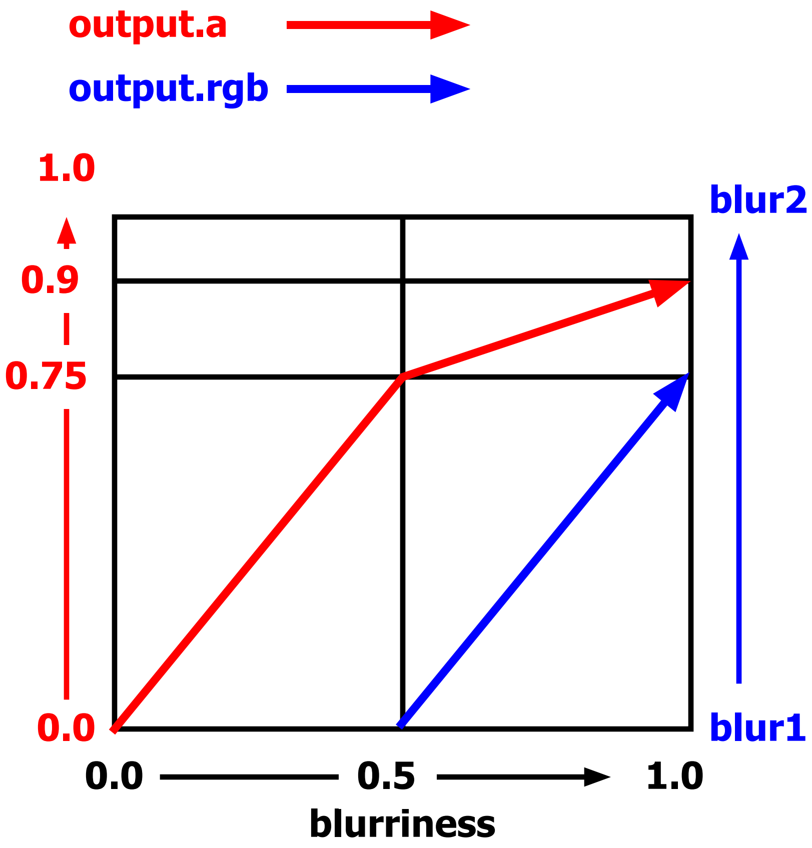
- Ideally, alpha and color outputs satisfy:
    - r : screen-space blurriness
    - blur1 : a bit blurred image (256x192)
    - blur2 : strongly blurred image (160x120)
  - When r = 0.0
      - .a = 0.0 (no blend)
      - .rgb = blur1 (will not affect the result)
    - The resulting pixel is 100% the frame buffer

Calculate Blend factor Alpha and Color (2)

- When r = 0.5
  - .a = somewhat smaller than 1.0
  - .rgb = blur1
  - The result is almost the blur1 image
- When r = 1.0
  - .a = slightly smaller than 1.0
  - .rgb = a blend of blur1 and blur2 (almost blur2)
  - The result is almost the blur2 image

DOF Shader Code

```
if (blurriness > 0.5) {
      out.a = blurriness * 0.25 + 0.75 ;
}
else {
      out.a = blurriness * 1.5 ;
}

// blurriness: 0.0 -> rgbFactor = -0.75
// blurriness: 0.5 -> rgbFactor = 0.0
// blurriness: 1.0 -> rgbFactor = 0.75
rgbFactor = blurriness * 0.75 - 0.75 ;

// lerp blur1 and blur2 by rgbFactor
out.rgb = blur1 + (blur2 - blur1) *
      rgbFactor ;
```



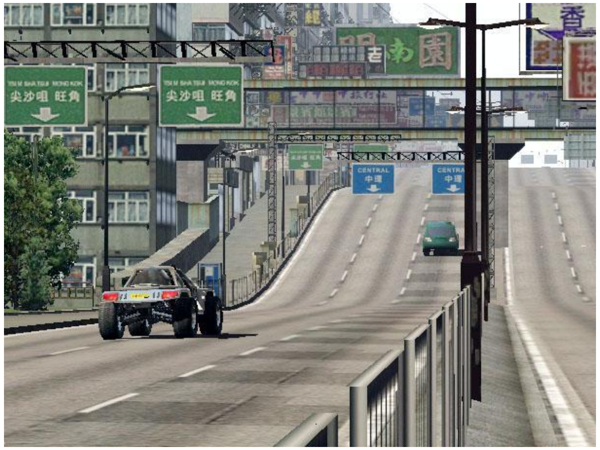
DOF Pixel Shader

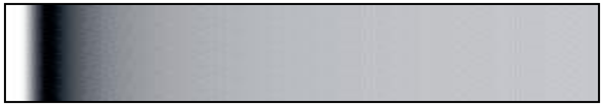
```
xps.1.1

def c0,  0.0f, 0.0f, 0.0f,  0.15f  // (0.9f - 0.75f)
def c1,  0.0f, 0.0f, 0.0f,  0.75f

tex t0                  // t0.a : W buffer (distance from camera)
texreg2ar t1, t0              // t1.a : screen-space blurriness
tex t2                  // t2   : blurred frame buffer (256x192)
tex t3                  // t3   : strongly blurred buffer (160x120)

mad_d2 r0.rgb,  t1_bx2.a, c0.a,  c1.a
+mov r0.a,  t1.a

mul r1.rgb,  t1_bx2.a, c1.a
+xmmc_x2 DISCARD.a, DISCARD.a, r0.a,  t1.a, c1.a,  1-ZERO, r0.b

// Color output will be alpha blended with the frame buffer based on the alpha output
xfc r1.b, t3, t2, ZERO, ZERO, ZERO, r0.a
```

- “DISCARD” output register discards the results

DOF processing textures

Original frame buffer

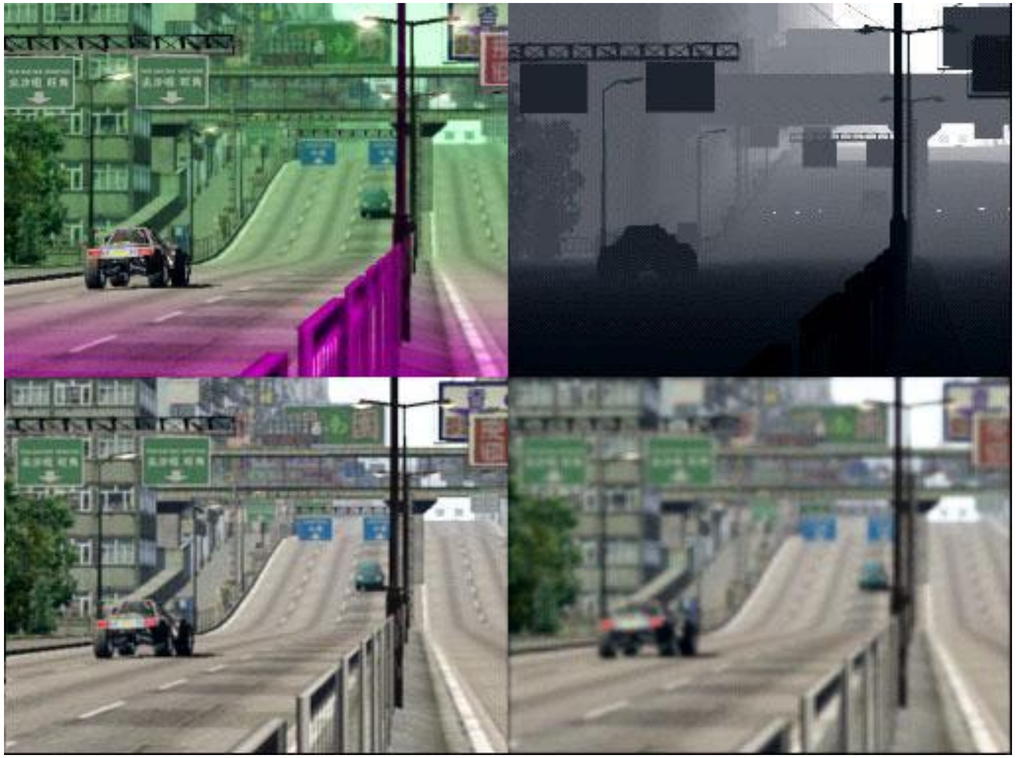




1D look-up texture that maps W value to blurriness

Final image with blurriness

W buffer



256x192 Blurred image

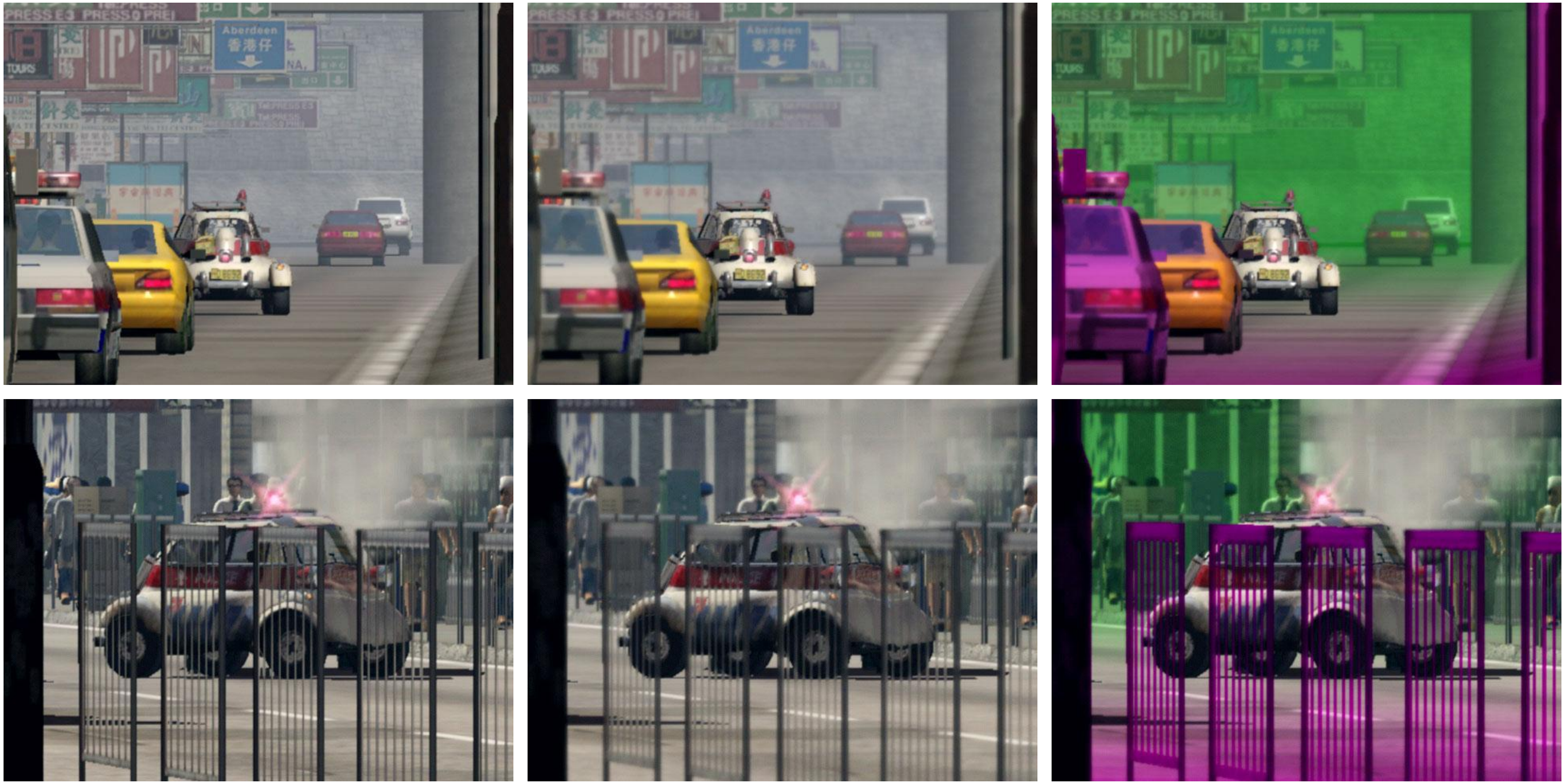
160x120 Strongly blurred

Depth of Field images

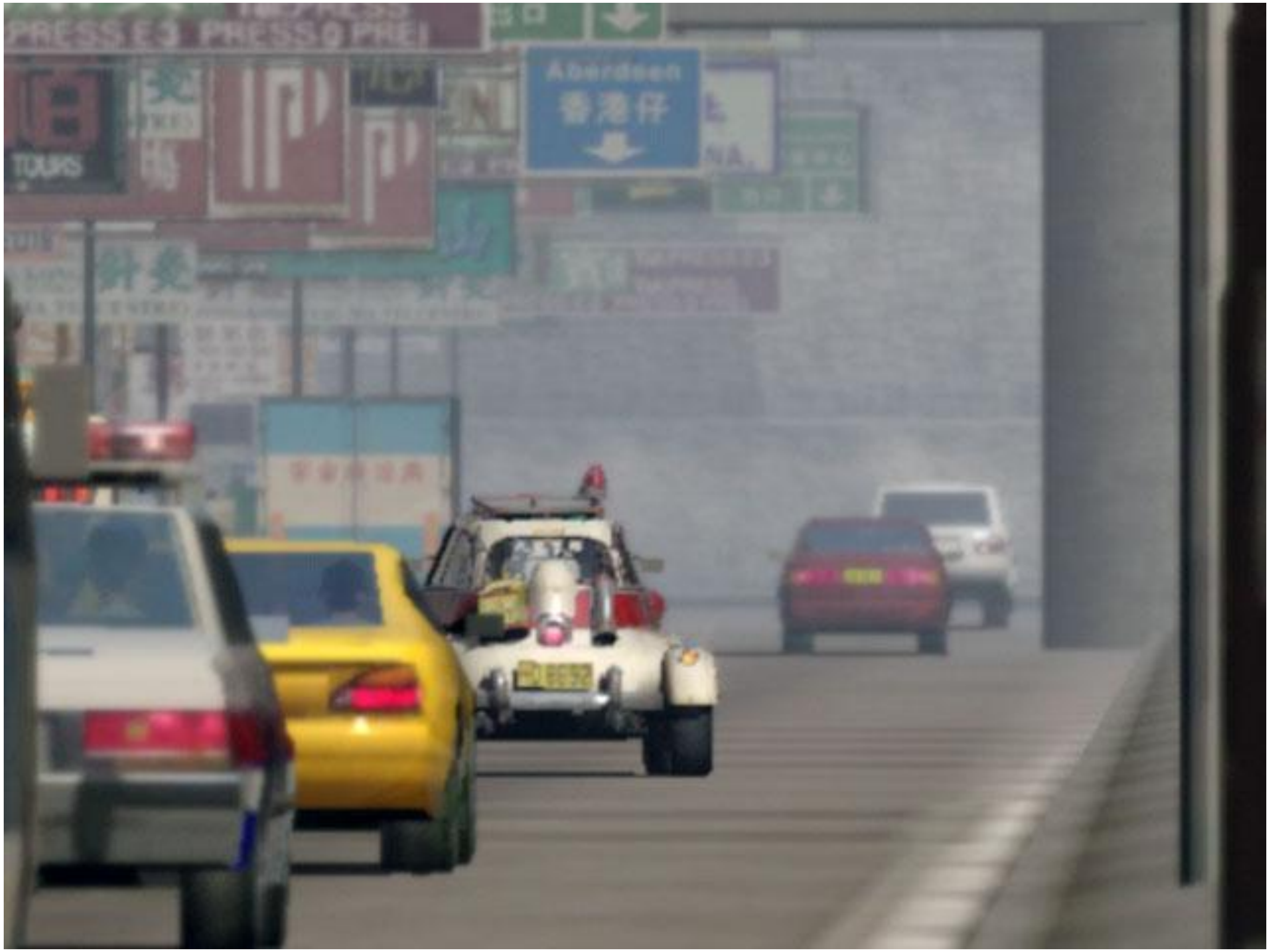
Original images

Final images

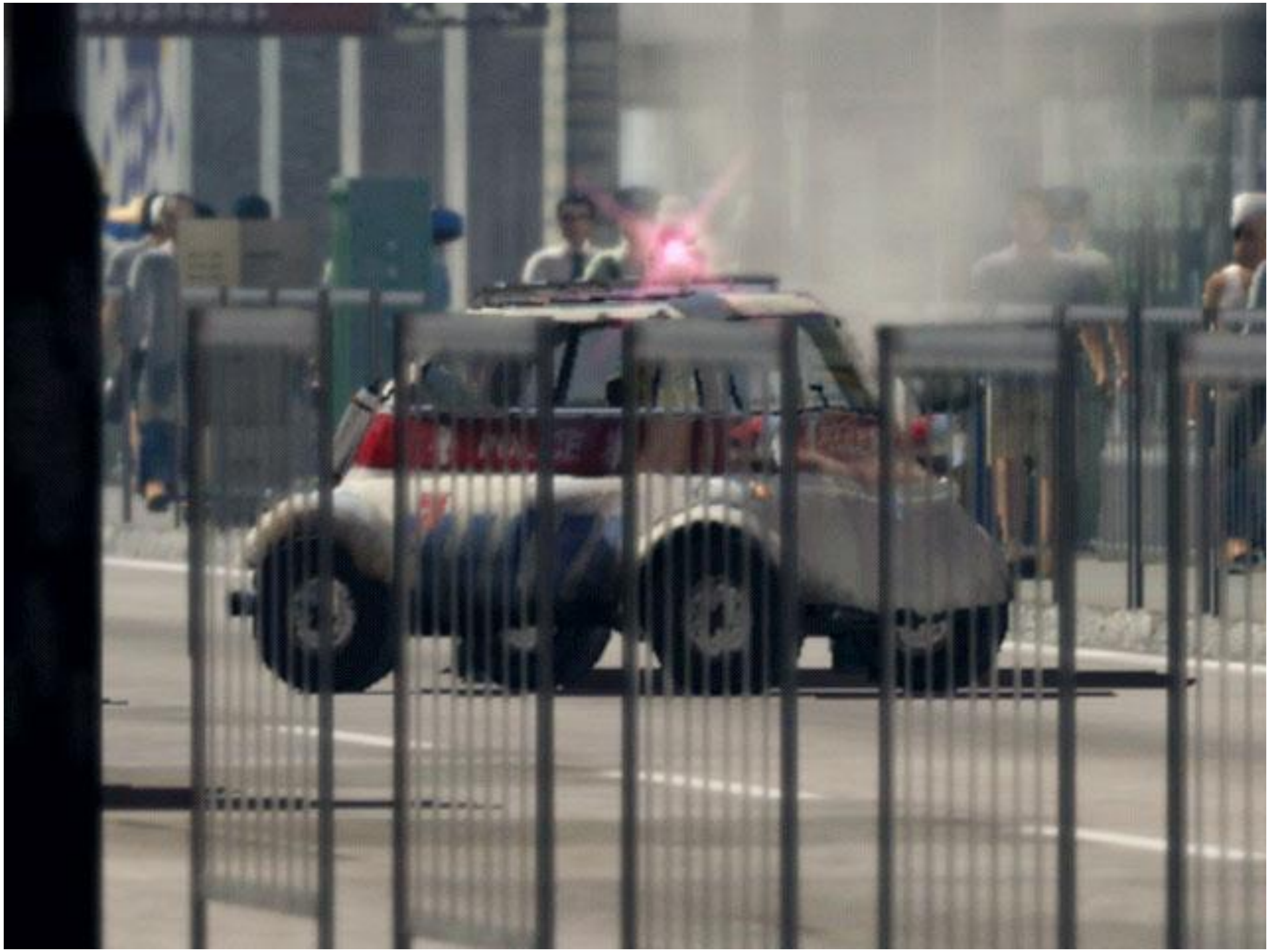
Final images with blurriness



Final Image (1)



Final Image (2)



Copyright© 2002 BUNKASHA PUBLISHING CO.,LTD.

Other post-processing filters

- Projector
  - Separation of RGB components
- Camera image
  - Emphasize contrast
  - Soften edges
- Illustration
  - Edge detection
  - Pale coloring

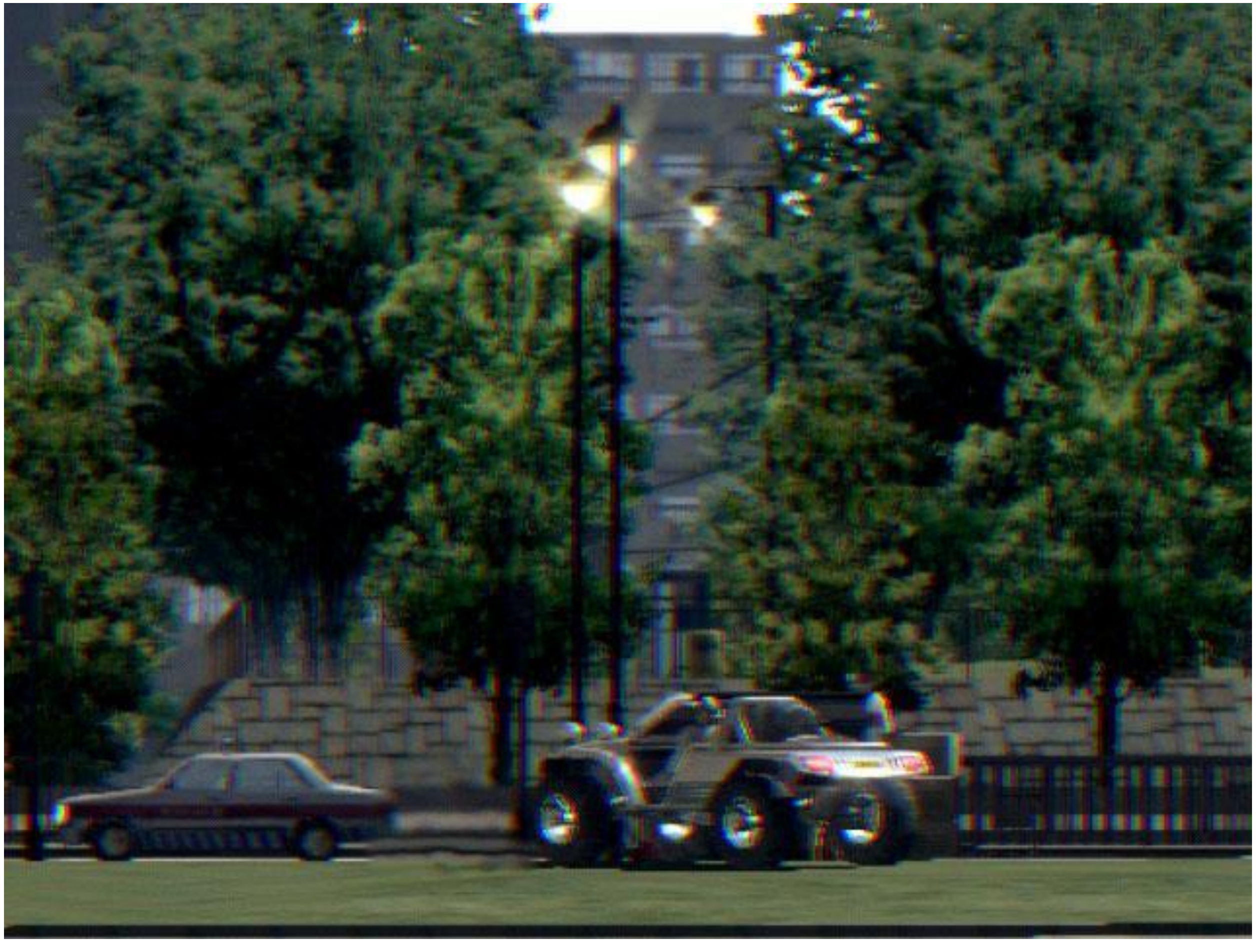
Projector (RGB separation) Pixel Shader

```
xps.1.1

//   c0.rgb : glare intensity
//   c2.rgb : fadeout color
//   c2.a   : fadeout factor
//   c3.rgb : modulator

def c5,  1.0f, 0.0f, 0.0f,  0.0f // R mask
def c6,  0.0f, 1.0f, 0.0f,  0.0f // G mask
def c7,  0.0f, 0.0f, 1.0f,  0.0f // B mask

tex t0// frame buffer(R jittered)
tex t1// frame buffer(G jittered)
tex t2// frame buffer(B jittered)
tex t3// generated glare
```

```
// Sum up using RGB masks
xmma DISCARD.rgb, DISCARD.rgb, r0.rgb, t0, c5,  t1, c6
mad r0.rgb,  t2, c7, r0

// Add glare
mad r0, t3, c0, r0

// Modulate and fadeout
xfc c2.a, c2, r0, ZERO, r0, c3, r0.a
```

Texcoords for t0,t1,t2 are jittered for RGB separation (Left/Center/Right)

Projector (RGB separation)



Copyright© 2002 BUNKASHA PUBLISHING CO.,LTD.

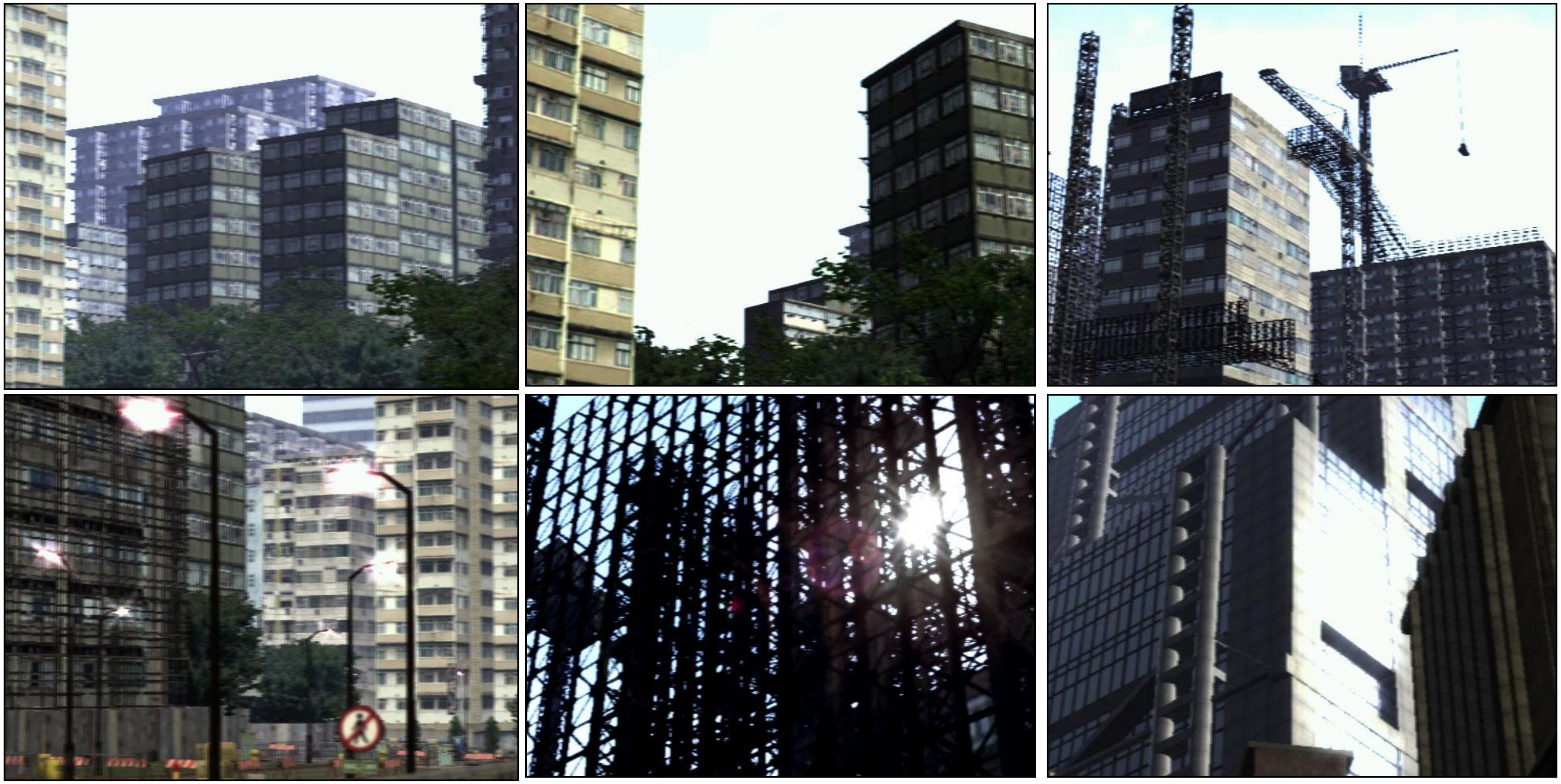
Camera image Pixel Shader

```
xps.1.1

//   c4.rgb : gray scale coefficients
def c4,  0.30f, 0.59f, 0.11f,  0.0f

// blend factor for three images
def c5,  0.0f, 0.0f, 0.0f,  0.5f
def c6,  0.0f, 0.0f, 0.0f,  0.333333333f

tex t0// frame buffer
tex t1// frame buffer
tex t2// frame buffer
tex t3// glare

// Soften frame buffer edges
lrp r0,  c5.a,  t1, t2
lrp r0,  c6.a,  t0, r0
```

```
// Add glare
mad r0,  t3, c0,  r0

// Calculate luminance
dp3 r1,  r0, c4
// Emphasize contrast
mul_x2  v0, r0, r0
lrp r0, r1,  r0, v0

// Modulate color
mul_x2 r0,  r0, c3
// Fadeout
xfc c2.a, c2, r0, ZERO, ZERO, ZERO, r0.a
```

Texcoords for t0,t1,t2 are jittered for softening edges

Camera image



Edge Detection Pixel Shader

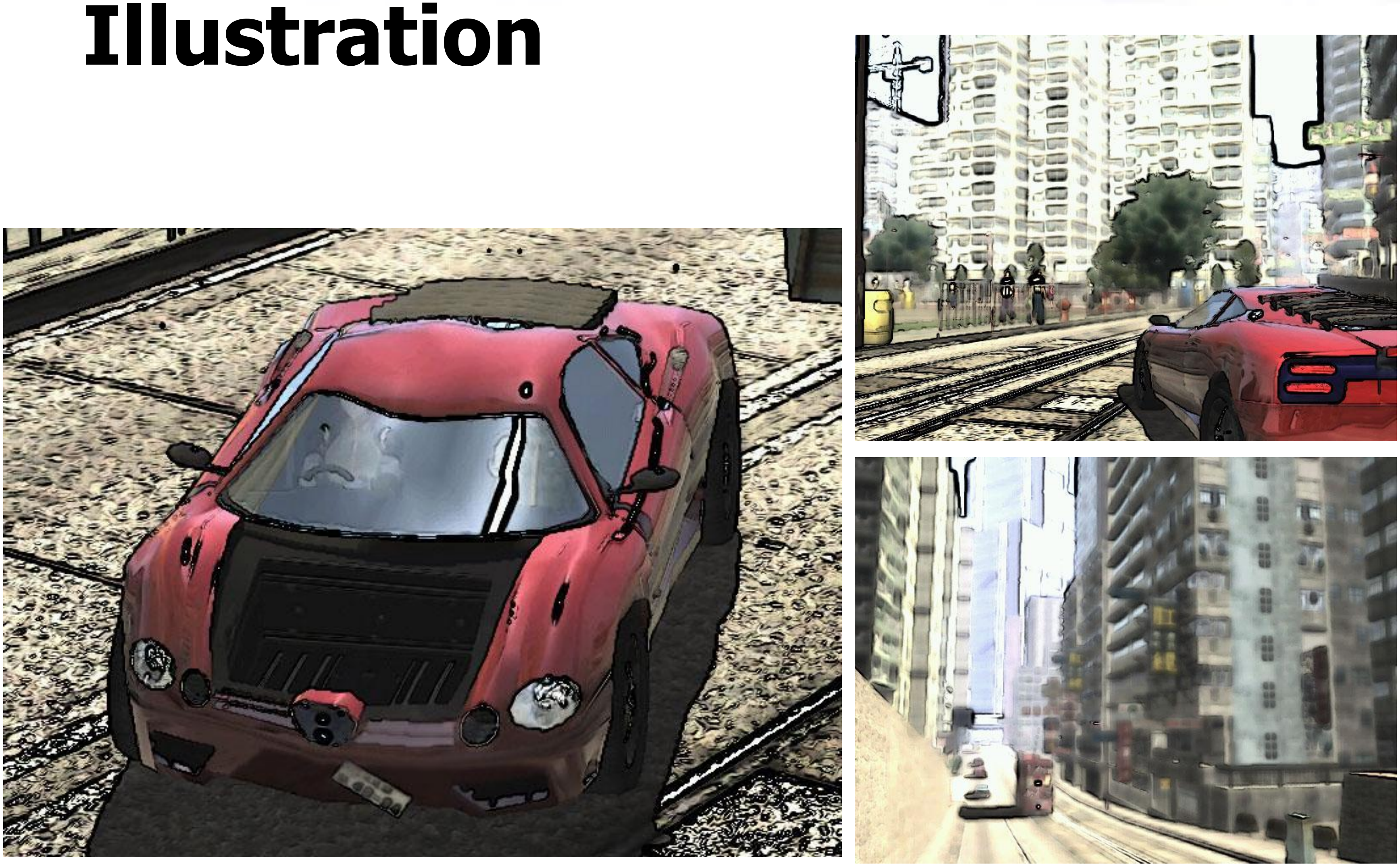
```
// Edge detection pixel shader
xps.1.1

// Up/Down/Left/Right jittered sampling
tex t0      // frame buffer (jittered)
tex t1      // frame buffer (jittered)
tex t2      // frame buffer (jittered)
tex t3      // frame buffer (jittered)

// R/G/B sub
sub_x2 r0,  t0, t1
sub_x2 r1,  t2, t3

// Approximate absolute values
// dp3_x4 r0,  r0, r0
// dp3_x4 r1,  r1, r1
xdd_x4 r0, r1,  r0, r0,  r1, r1

// complement
sub r0, 1-r0, r1 // 1 - r0 - r1
```

Illustration Pixel Shader

```
// Illustration pixel shader
xps.1.1

def c2,  0.0f, 0.0f, 0.0f,  0.0f
def c3,  0.0f, 0.0f, 0.0f,  0.5f
def c4,  0.30f, 0.59f, 0.11f,  0.0f
def c7,  0.0f, 0.0f, 0.5f,  0.75f

// jittered sampling
tex t0      // frame buffer (jittered)
tex t1      // frame buffer (jittered)
tex t2      // frame buffer (jittered)
tex t3      // frame buffer (jittered)
```

```
// Edge detection
sub_x2 r0,  t0, t1
sub_x2 r1,  t2, t3
xdd_x4 r0, r1,  r0, r0,  r1, r1
sub r0, 1-r0, r1 // 1 - r0 - r1

// Pale coloring
dp3_x4 t1, t0, c4
lrp t1.a,  c7.b,  1-ZERO, t1.a
lrp t0, c7.a, t0, t1.a
mul_x2 r0, r0_bx2, t0

xfc c2.a, c2, PROD, ZERO, r0, r0, r0.a
```

Illustration



Images Copyright© 2002 BUNKASHA PUBLISHING CO.,LTD.