SCHOOL of ELECTRICAL ENGINEERING & COMPUTING
FACULTY of ENGINEERING & BUILT ENVIRONMENT
The UNIVERSITY of NEWCASTLE

Comp3320/6370 Computer Graphics

Course Coordinator: Associate Professor Stephan Chalup

Semester 2, 2018

LECTURE w04

Effects

August 19, 2018

OVERVIEW

Note: These lecture slides use concepts, statements and figures from the books Möller and Haines (1999), Akenine-Möller et al. (2018), Shirley (2009) and some other sources listed in the literature section at the end.

Special Effects

- *Special effects* is a relative term in computer graphics.
- Some time ago image morphing or even Gouraud shading was regarded as *special*.
- Availability of graphic accelerators $\leadsto$ eg. breaking up the lighting equation into pieces.
- New capabilities become available in hardware $\leadsto$ new special effects become possible.
- Example: Reflections and shadows can be performed more rapidly when the *stencil buffer* is available.

Buffers

- The *colour-buffer:*
  - A rectangular array of colours (a red, a green and a blue component for each colour).
  - Can have different colour modes:
    - *Indexed or pseudo colour:* 1 byte per pixel, allowing 256 colours.
    - *High colour:* 2 bytes per pixel, of which 15 or 16 bits are used for the colour, giving 32,768 or 65,536 colours, resp..
    - *True colour or RGB colour:* 3–4 bytes per pixel, 24 bits are used for the colour, giving about 16.8 million colours. A 32 bit representation can be present for speed purposes. The extra 8 bits for the alpha channel.
  - Some systems offer the option of mixing modes. Many other modes exist.

Buffers

- The *z-buffer or depth-buffer:*
  - Manages depth information and visibility in 3D graphics.
  - (x,y)-array of the same size as the colour buffer.
  - Typically 16–32 bits per pixel.
  - For each pixel it stores the z-value from the camera to the currently closest primitive.
  - z-buffer algorithm (typically in graphics card)
    * z-buffer stores z-value, framebuffer stores the colour value.
    * If point rendered at (x,y) is no farther from the viewer than is the point whose colour and depth are currently in the buffer, then the new point's colour and depth replace the old values. I.e. a close object hides a farther one.

Buffers

- The *frame-buffer:*
  - Generally consists of all buffers of a system.
  - Sometimes used to mean just the colour and the z-buffer as a set.

Buffers

- The *accumulation buffer:*
  - has the same resolution and more bit depth than the desired image.
  - Typically twice as many bits as the colour buffer.
  - Accumulation buffers are part of OpenGL and some hardware systems support their use.
  - Images are summed up in the accumulation buffer. After rendering, the image is averaged and sent to the display.
  - Adds and subtracts images from the colour buffer.
  - Can be used for motion blur, depth of field, antialiasing and soft shadows (Example: Simulate an area light by sampling a number of point lights on its surface; for each point light render an image and add it to the accumulation buffer; the average image has soft shadows.)

Buffers

- The *A-buffer or multisampling:*
  - Limitation of the z-buffer is that only one object is stored per pixel. If several transparent objects overlap use the A-buffer.
  - It can be used to increase the sampling rate.
  - Used to generate high-quality renderings at non-interactive speed.
  - A polygon's approximate coverage for each grid cell is calculated.

Buffers

- The $alpha$-*channel:*
  - Associated with the colour buffer.
  - Stores a related opacity value for each pixel.
  - RGBα
  - Compare *chroma-keying* or *blue-screen matting*: A particular colour is designated to be considered transparent; where it is detected the background is displayed.
    (ie. only either completely transparent or opaque and no intermediate α values possible.)

Buffers

- The *stereo buffer:*
  - Some hardware has support for stereo rendering.
  - Use two images (*stereo pair*) to make object look 3D.
  - *Stereo vision:* Two eyes $\leadsto$ two images $\leadsto$ visual system retrieves depth information.
  - red-green glasses or head-mounted displays or shutter glasses.

Buffers

- The *stencil buffer:*
  - A special buffer that is not displayed.
  - Normally used to mask off areas of the screen and to make them unwritable.
  - Part of OpenGL.
  - Requires same resolution as the colour buffer.
  - Usually contains 1–8 bits per pixel.
  - Can be used to control rendering into the colour and z-buffer.
  - Reflections and shadows can be performed more rapidly when the stencil buffer is available.
  - A powerful tool for creating special effects.

Transparency

- Limited and simplistic effects are available for real-time rendering.
- Normally unavailable:
  - Refraction (bending of light)
  - Attenuation due to thickness of the transparent object.
  - Reflectivity and transmission changes due to the viewing angle.
- Normally available:
  - Render a surface as semi-transparent.
  - Blending a surface's colour with the object behind it.

Screendoor Transparency

- Idea: Render the transparent polygon with a checkerboard fill pattern.
- Problems:
  - Object can be only 50% transparent.
  - Only one transparent object can be convincingly rendered.
- Advantage: simplicity, no special hardware support required.

Alpha Blending

- Concept: Blend the colour of the object in the background with the transparent object's colour.
- Usually with each pixel an RGB colour value and a Z-buffer depth are associated.
- An optional α value can be stored for each pixel to determine the degree of opacity.

Alpha Blending, Cont.

- Idea: RGBα, where α is a value describing the opacity of an object for that given pixel.
  - $\alpha = 1.0$ means the object is opaque and entirely covers the pixel's area of interest.
  - $\alpha = 0.0$ means the pixel is not obscured at all.
- To make an object transparent it is rendered on top of the existing scene with $\alpha \leq 1.0$.
- Each pixel covered by the object receives a RGBα.
- Two ways to store the RGBα values:
  - *pre-multiplied* alphas: RGB values are multiplied by the alpha value before being stored.
  - *un-multiplied* alphas

The *over* Operator

- For generating a transparent object it is rendered on top of the scene with an $\alpha < 1$.
- The *over* operator:

$$\mathbf{c}_0 = \alpha \mathbf{c}_s + (1 - \alpha)\mathbf{c}_d$$

  where
  - $\mathbf{c}_s$ = colour of the transparent object (called *source*).
  - $\mathbf{c}_d$ = pixel colour before blending (called the *destination*).
  - $\mathbf{c}_0$ = resulting colour due to placing the transparent object *over* the existing scene.
- Completely transparent: $\alpha = 0$.

Alternatives to the *over* Operator

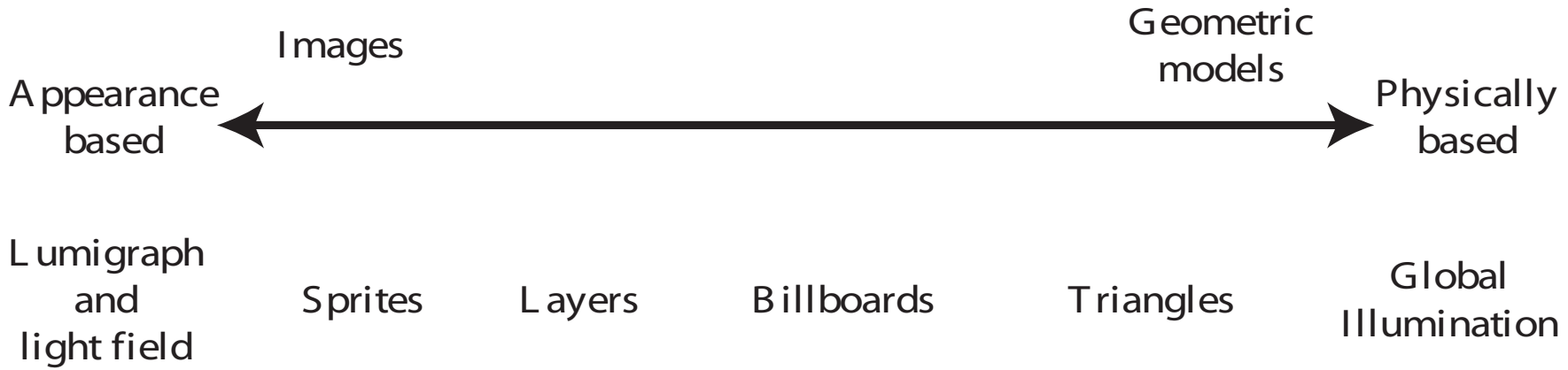
- To render transparent object properly into a scene usually requires sorting: First the opaque objects are rendered and the the transparent objects are blended on top of them in back-to-front order.
- A method that does not require sorting:

$$\mathbf{c}_0 = \alpha \mathbf{c}_s + \mathbf{c}_d$$

  where
  - $\mathbf{c}_s$ = colour of the transparent object (called *source*).
  - $\mathbf{c}_d$ = pixel colour before blending (called the *destination*).
  - $\mathbf{c}_0$ = resulting colour due to placing the transparent object *over* the existing scene.

- There is no *correct way* to render a scene.

- Representing objects with polygons is not the only way to get them to the screen.
- Speed is important.
- *Level-of-detail techniques*: Use simpler versions of an object (less triangles) as it makes less contribution to an image.
- Use images instead of polygons $\rightsquigarrow$ *image-based rendering*.
- Radiosity or ray tracing are only feasible on high end machines.
- The rendering spectrum goes from physically based geometric models to appearance based images.


Images
Geometric models
Appearance based
Physically based
Lumigraph and light field
Sprites
Layers
Billboards
Triangles
Global Illumination

Image–Based Rendering

- *Sprites*
  - One of the simplest image-based primitives.
  - It is an image that moves around on the screen.
  - Eg. mouse cursor.
  - Can be zoomed by integer factors.
  - Different view angles can use different sprites.
  - However, we have a weak illusion due to jumps between sprites.
- Think of a scene as a series of sprite layers (no z-buffer required)
  - Treat layers independently.
  - Can use separate shadow and reflection sprite layers.
- Interpenetrating objects can be treated as one sprite.

Quicktime VR

- *Quicktime VR*
  - A 360^o panoramic image surrounds the viewer as a cylindrical image.
  - As the camera's orientation changes, the proper part is displayed.
  - Limited to a single location.
  - Better than static scene because viewer's head can turn and tilt.
  - Good for scenes in buildings or streets.

Lumigraph

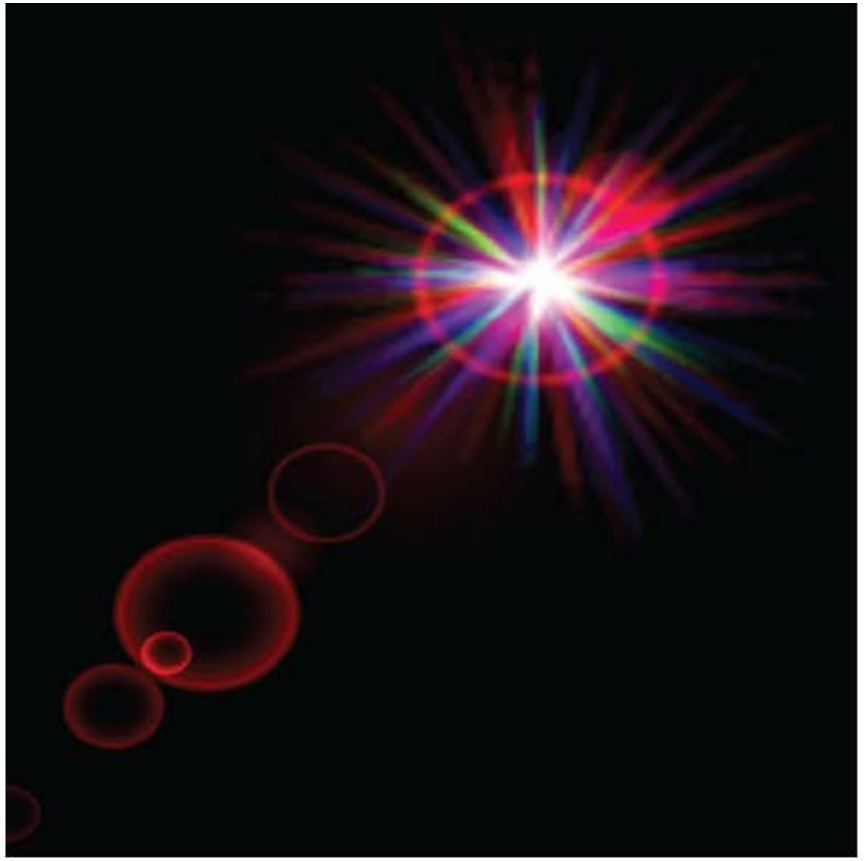
- *Lumigraph*
  - A single object is viewed from a set of viewpoints.
  - Interpolation between stored views $\rightsquigarrow$ create a new view.
  - Compare holography: 2D array of views captures the object.
  - Can capture a real object and redisplay it from any angle (!)

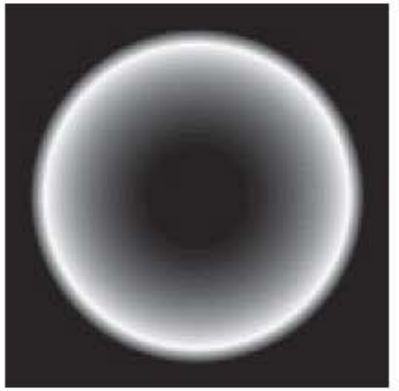
Glare Effects

- *Lens Flare*
  - Caused by the lens when directed at bright light.
  - Consists of a *halo* and a *ciliary corona*.
  - Prism effect $\rightsquigarrow$ halo $=$ ring around the light
  - Density fluctuations within the lens $\rightsquigarrow$ ciliary corona $=$ rays radiating from a point
  - An animated ciliary corona can create sparkle effects.
- *Bloom*
  - Scattering in the lens $\rightsquigarrow$ bloom $=$ glow around the light $+$ dimming contrast elsewhere in the scene.

These effects are associated with brightness. They are added to digital images.

Glare Effects





Billboarding

- *Billboard* = textured polygon that is dynamically rotated for best visibility.
- *Billboarding* = orienting a billboard based on the view direction.
- Using billboards together with alpha blending and animation phenomena that have not solid surfaces can be modelled: Fire, smoke, dust, explosions, vapor trails, clouds, energy beams, vegetation ...
- There are different forms of billboards, eg.:
  - Screen–Aligned Billboards
  - World–Aligned Billboards
  - Axial Billboards

Screen–Aligned Billboards

- Align billboard with the screen and maintain a constant up vector.
- Ensures that displayed text on the billboard is always displayed upright. Can also be used for effects such as lens flares.
- How to obtain the transform (basic but slow version):
  - The polygon's surface normal $\mathbf{n}$ should be aligned with the negation of the view direction $-\mathbf{v}_{dir}$. (To align use vector–vector rotation around cross product)
  - Similarly align the polygon's original up–direction $\mathbf{u}$ with the up–direction for the viewer $\mathbf{v}_{up}$.
  - To obtain the transform concatenate the corresponding two matrices.

Screen–Aligned Billboards (faster version)

- How to screen-align a billboard faster:

  Let $\mathbf{M} = [\mathbf{n}\ \mathbf{u}\ \mathbf{n} \times \mathbf{u}]$ be the matrix with $\mathbf{n}$, $\mathbf{u}$ and $\mathbf{n} \times \mathbf{u}$ as column vectors and
  similarly define $\mathbf{N} = [-\mathbf{v}_{dir}\ \mathbf{v}_{up}\ -\mathbf{v}_{dir} \times \mathbf{v}_{up}]$.

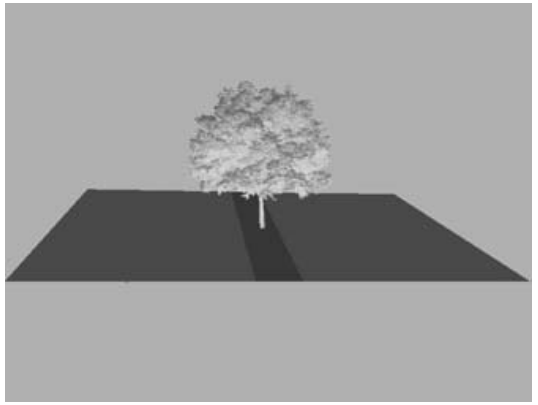
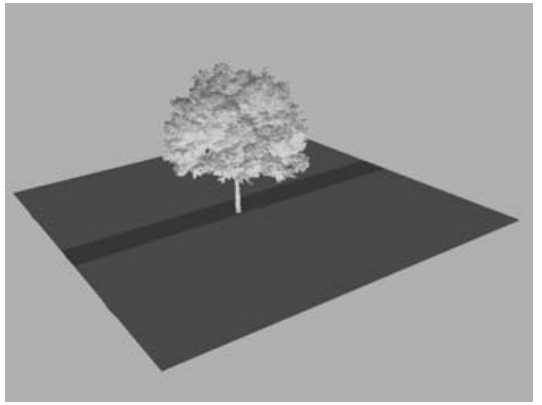
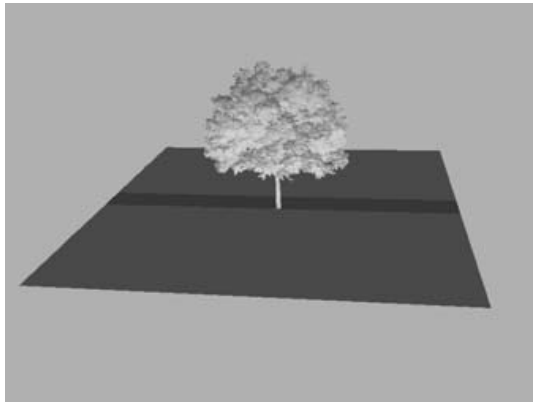
  The billboard matrix is then $\mathbf{N}\mathbf{M}^{\mathbf{T}}$ if we assume that all vectors have unit length.

  Note: Details to understand this item will be addressed later in lecture transforms II where we look at coordinate transforms. We will see that $\mathbf{M}$ and $\mathbf{N}$ are both elements of the orthogonal group and therefore their inverse equals their transpose.

(Important for exams.)

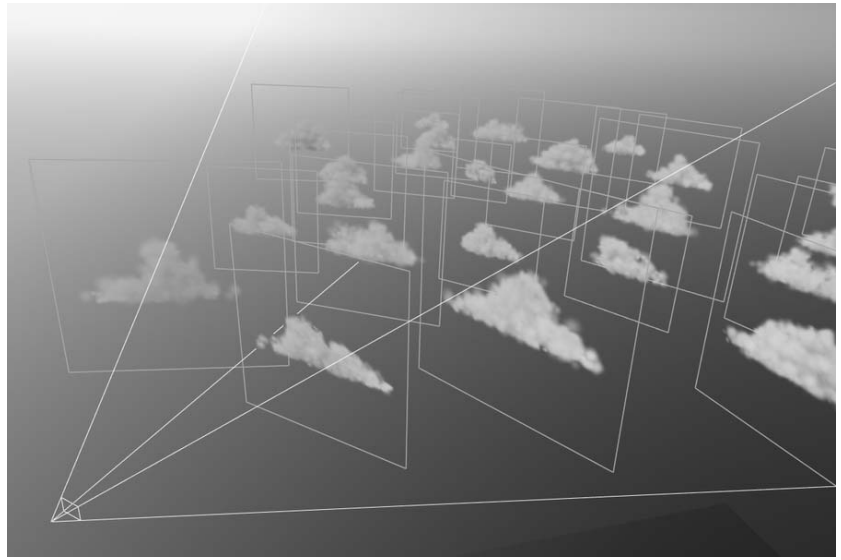
Axial Billboards

- The textured object does not normally face straight-on towards the viewer.
- It is allowed to rotate around some fixed world-space axis.
- Aligns itself so as to face the viewer as much as possible.
- Can be used to display objects such as trees (where the up-vector is then along the trunk):

Impostors

- Billboard that is created on the fly by rendering a complex object from the current viewpoint into an image texture. This is then mapped onto a billboard.
- Work best for static distant objects.
- Purpose is to gain speed.
- Sometimes impostors are called sprites.



Full-Screen Billboarding

- A screen–aligned billboard that covers the whole screen.
- Can place the billboard in front or behind everything in the scene.
- Blend a green billboard on top of the whole scene $\rightsquigarrow$ feel like viewing the scene through night goggles.
- Flash effect: increase and decrease a foreground billboard in brightness over time.
- Example: Sky background for a driving simulator.

Exercise 14 (Homogeneous and non-homogeneous coordinates)

Question: Below several points (or vectors ?) are given in homogeneous coordinates. Determine for each of them the corresponding non-homogeneous 3D Cartesian coordinates.

a) (3, 6, 5, 1)
b) (2, 4, 6, 4)
c) (0, 0, 2, 0.25)
d) (0, 0, 0, 1)
e) (1, 0, 0, 0)

Exercise 15 (Inverse of some transform) Question: Let $\mathbf{F} \in M(4 \times 4, \mathbf{R})$ have the general form of a 3D transform in homogeneous coordinates. That is, if the submatrix $\mathbf{M} \in M(3 \times 3, \mathbf{R})$ is either a 3D rotation, scaling or shearing matrix and the submatrix $\mathbf{T} \in M(3 \times 1, \mathbf{R})$ is a translation vector then $\mathbf{F}$ can be written as:

$$\mathbf{F} = \left[\begin{array}{c|c} \mathbf{M} & \mathbf{T} \\ \hline \mathbf{0} & 1 \end{array}\right] = \left[\begin{array}{ccc|c} m_{00} & m_{01} & m_{02} & t_x \\ m_{10} & m_{11} & m_{12} & t_y \\ m_{20} & m_{21} & m_{22} & t_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

Let $\mathbf{N} = \mathbf{M}^{-1}$ be the inverse of $\mathbf{M}$. Show that the inverse of $\mathbf{F}$ is given by:

$$\mathbf{F}^{-1} = \left[\begin{array}{c|c} \mathbf{N} & -\mathbf{NT} \\ \hline \mathbf{0} & 1 \end{array}\right] = \left[\begin{array}{ccc|c} n_{00} & n_{01} & n_{02} & -(\mathbf{NT})_x \\ n_{10} & n_{11} & n_{12} & -(\mathbf{NT})_y \\ n_{20} & n_{21} & n_{22} & -(\mathbf{NT})_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

LITERATURE

Akenine-Möller, T. and Haines, E. (2002). *Real-Time Rendering*. A K Peters, second edition.

Akenine-Möller, T., Haines, E., and Hoffman, N. (2008). *Real-Time Rendering*. A K Peters, third edition.

Akenine-Möller, T., Haines, E., Hoffman, N., Pesce, A., Hillaire, S., and Iwanicki, M. (2018). *Real-Time Rendering*. A K Peters/CRC Press, fourth edition.

Foley, J., vanDam, A., Feiner, S., and Hughes, J. (1997). *Computer Graphics: Principles and Practice, Second Edition in C*. Addison–Wesley.

Foley, J. D., vanDam, A., Feiner, S. K., Hughes, J. F., and Phillips, R. L. (1994). *Introduction to Computer Graphics*. Addison–Wesley.

Hill, F. (2001). *Computer Graphics Using OpenGL*. Prentice Hall, second edition.

Hill, F. S. and Kelley, S. M. (2007). *Computer Graphics Using OpenGL*. Prentice Hall, third edition.

Möller, T. and Haines, E. (1999). *Real-Time Rendering*. A K Peters.

Shirley, P. (2009). *Fundamentals of Computer Graphics*. A K Peters, third edition.