Lecture 16 of 41

Transparency, Painter's Algorithm, & *Z*-Buffer
Lab 3a: Shading & Transparency

William H. Hsu

Department of Computing and Information Sciences, KSU

KSOL course pages: http://bit.ly/hGvXlH / http://bit.ly/eVizrE
Public mirror web site: http://www.kddresearch.org/Courses/CIS636
Instructor home page: http://www.cis.ksu.edu/~bhsu

Readings:

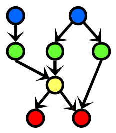
Today: §2.6, 20.1 Eberly *2e*

OpenGL: A Primer on shading, alpha blending

Khronos Group docs on transparency: **http://bit.ly/hRaQgk**

Wikipedia, *Painter's Algorithm*: **http://bit.ly/eeebCN**

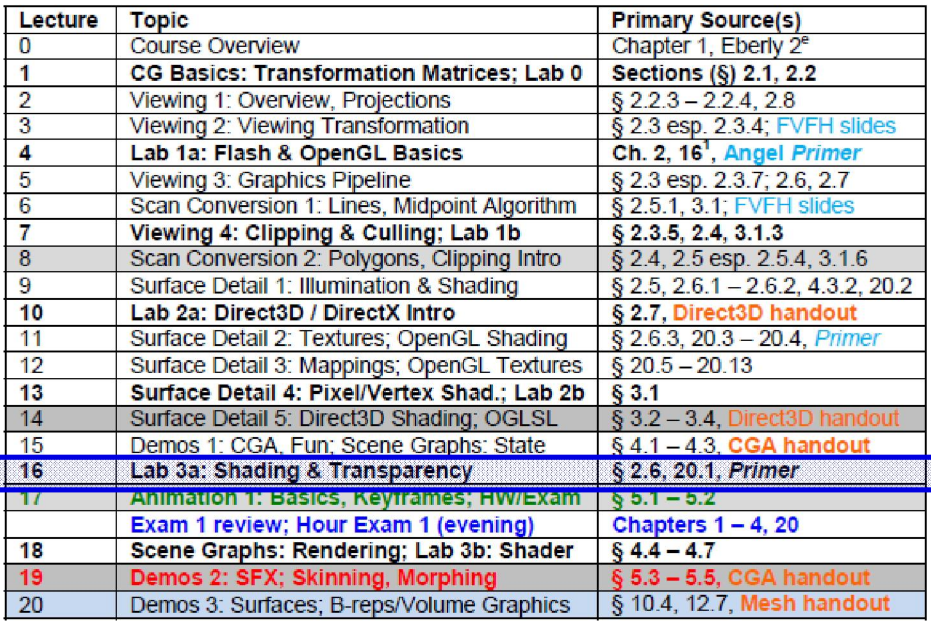
Wikipedia, *Z-buffering*: **http://bit.ly/gGRFMA**



Lecture Outline

- Reading for Last Class: §4.1 – 4.3, Eberly *2e*; CGA handout
- Reading for Today: §2.6, 20.1, Eberly *2e*; OpenGL primer material
- Reading for Next Class: §5.1 – 5.2, Eberly *2e*
- Last Time: Scene Graphs; CGA Demos, Videos
  - Scene graphs and state – main topic
  - State of CGA: videos and discussion
  - Demos to download
    - Adobe Maya: http://students.autodesk.com
    - NewTek Lightwave: http://www.newtek.com/lightwave/
- Today: Shading and Transparency in OpenGL
  - Transparency revisited
  - OpenGL how-to: http://bit.ly/hRaQgk
    - Alpha blending (15.020, 15.040)
    - Screen-door transparency (15.030)
  - Painter's algorithm & depth buffering (z-buffering)

Where We Are



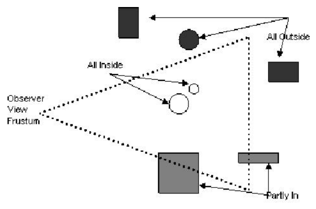
Lecture	Topic	Primary Source(s)
0	Course Overview	Chapter 1, Eberly 2e
1	**CG Basics: Transformation Matrices; Lab 0**	**Sections (§) 2.1, 2.2**
2	Viewing 1: Overview, Projections	§ 2.2.3 – 2.2.4, 2.8
3	Viewing 2: Viewing Transformation	§ 2.3 esp. 2.3.4; FVFH slides
4	**Lab 1a: Flash & OpenGL Basics**	**Ch. 2, 16[1], Angel *Primer***
5	Viewing 3: Graphics Pipeline	§ 2.3 esp. 2.3.7; 2.6, 2.7
6	Scan Conversion 1: Lines, Midpoint Algorithm	§ 2.5.1, 3.1; FVFH slides
7	**Viewing 4: Clipping & Culling; Lab 1b**	**§ 2.3.5, 2.4, 3.1.3**
8	Scan Conversion 2: Polygons, Clipping Intro	§ 2.4, 2.5 esp. 2.5.4, 3.1.6
9	Surface Detail 1: Illumination & Shading	§ 2.5, 2.6.1 – 2.6.2, 4.3.2, 20.2
10	**Lab 2a: Direct3D / DirectX Intro**	**§ 2.7, Direct3D handout**
11	Surface Detail 2: Textures; OpenGL Shading	§ 2.6.3, 20.3 – 20.4, *Primer*
12	Surface Detail 3: Mappings; OpenGL Textures	§ 20.5 – 20.13
13	**Surface Detail 4: Pixel/Vertex Shad.; Lab 2b**	**§ 3.1**
14	Surface Detail 5: Direct3D Shading; OGLSL	§ 3.2 – 3.4, Direct3D handout
15	Demos 1: CGA, Fun; Scene Graphs: State	§ 4.1 – 4.3, **CGA handout**
16	**Lab 3a: Shading & Transparency**	**§ 2.6, 20.1, *Primer***
17	**Animation 1: Basics, Keyframes; HW/Exam**	**§ 5.1 – 5.2**
	Exam 1 review; Hour Exam 1 (evening)	**Chapters 1 – 4, 20**
18	**Scene Graphs: Rendering; Lab 3b: Shader**	**§ 4.4 – 4.7**
19	**Demos 2: SFX; Skinning, Morphing**	**§ 5.3 – 5.5, CGA handout**
20	Demos 3: Surfaces; B-reps/Volume Graphics	§ 10.4, 12.7, **Mesh handout**

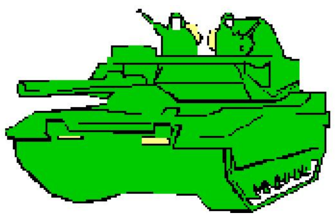
Lightly-shaded entries denote the due date of a written problem set; heavily-shaded entries, that of a machine problem (programming assignment); blue-shaded entries, that of a paper review; and the green-shaded entry, that of the term project.

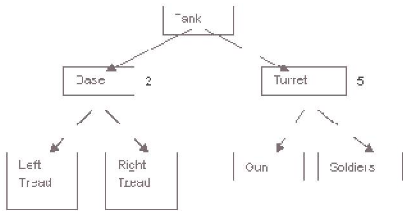
Green, **blue** and **red** letters denote **exam review**, **exam**, and **exam solution review** dates.

Review [1]: Scene Graphs

- Scene Graph: General Data Structure used in CG
  - Used to: compute visibility, set up rendering pipeline
  - Actual graph: general graph, forest, or rooted tree
- Scene Graph Traversal: Initial Step – Drives Rendering
- Features of Scene Graphs
  - Spatial partitioning: *e.g.*, using bounding volume hierarchies
  - Leaves: primitive components
  - Interior nodes: assembly operations, modelview transformations
  - Root(s): scene or major objects



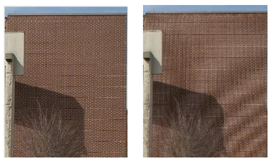




Review [2]: Aesthetic Considerations

- Non-Photorealistic Rendering: Aimed at Achieving Natural Aesthetic
  - Cartoon shaders: use sharp gradient (thresholded)
  - Pencil shaders: blurring, stippling
- CGA and Realism
- Aliasing (see Wikipedia: http://bit.ly/flkCkr)
  - Term from signal processing
  - Two sampled signals indistinguishable from (aliases of) one another
  - Examples: jaggies, Moiré vibration (Moiré pattern)
  - Anti-aliasing: operations to prevent such effects
- Temporal Aliasing
  - Similar effect in animation
  - Small artifact can be much more jarring!
  - Example: think of flecks in traditional film reels

© 2004 – 2009 Wikipedia, *Jaggies*
http://bit.ly/flkCkr


© 2004 – 2009 Wikipedia, *Aliasing*
http://bit.ly/flkCkr

Review[3]: CG Feature Films & Shorts

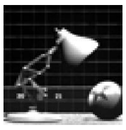
Monsters Inc. 2
© 2012 Disney/Pixar
http://youtu.be/cJHU9IYvWUg

Toy Story 3
© 2010 Disney/Pixar
http://youtu.be/JcpWXaA2qeg

Tron: Legacy
© 2010 Walt Disney Pictures
http://youtu.be/plwXwVJZ3BY

Kung-Fu Panda
© 2008 DreamWorks Animation SKG
http://bit.ly/h8krLv

Happy Feet
© 2006
Warner Brothers
http://bit.ly/gTnp2V

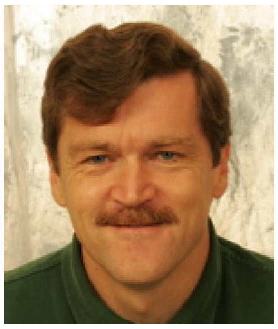


Luxo Jr.
© 1986 Pixar Animation Studios
http://youtu.be/L_oL_27KqgU

Shrek Forever After
© 2010 DreamWorks Animation SKG
http://youtu.be/u7__TG7swg0

Wall-E
© 2008 Disney/Pixar
http://bit.ly/eKDwkk

Review [4]: OpenGL Shading (Overview)



Frank Pfenning
Professor of Computer Science
School of Computer Science
Carnegie Mellon University
http://www.cs.cmu.edu/~fp/

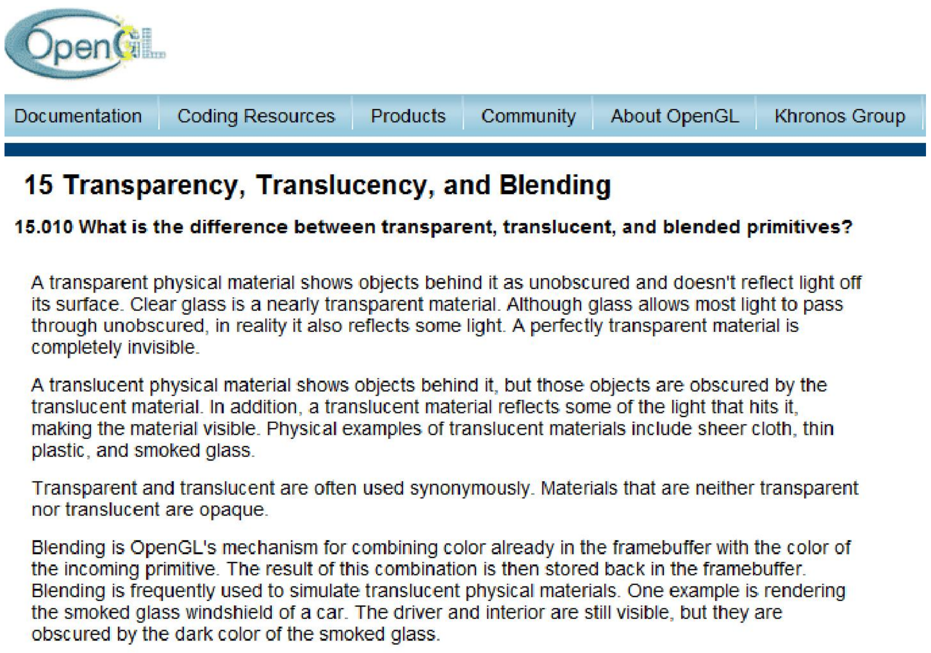
See also: *OpenGL: A Primer, 3^e* (Angel)
http://bit.ly/hVcVWN

- **Set Up Point Light Sources**
  - Directional light given by "position" vector
    ```
    GLfloat light_position[] = {-1.0, 1.0, -1.0, 0.0};
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    ```
  - Point source given by "position" point
    ```
    GLfloat light_position[] = {-1.0, 1.0, -1.0, 1.0};
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    ```
- **Set Up Materials, Turn Lights On**
  ```
  GLfloat mat_specular[]={0.0, 0.0, 0.0, 1.0};
  GLfloat mat_diffuse[]={0.8, 0.6, 0.4, 1.0};
  GLfloat mat_ambient[]={0.8, 0.6, 0.4, 1.0};
  GLfloat mat_shininess={20.0};
  glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
  glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
  glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
  glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess);

  glShadeModel(GL_SMOOTH); /*enable smooth shading */
  glEnable(GL_LIGHTING); /* enable lighting */
  glEnable(GL_LIGHT0);  /* enable light 0 */
  ```
- **Start Drawing (`glBegin` … `glEnd`)**

Adapted from slides © 2003 F. Pfenning, Carnegie Mellon University
http://bit.ly/g1J2nj

Transparency in OpenGL [1]: Transparent vs. Translucent, Blended



15 Transparency, Translucency, and Blending

15.010 What is the difference between transparent, translucent, and blended primitives?

A transparent physical material shows objects behind it as unobscured and doesn't reflect light off its surface. Clear glass is a nearly transparent material. Although glass allows most light to pass through unobscured, in reality it also reflects some light. A perfectly transparent material is completely invisible.

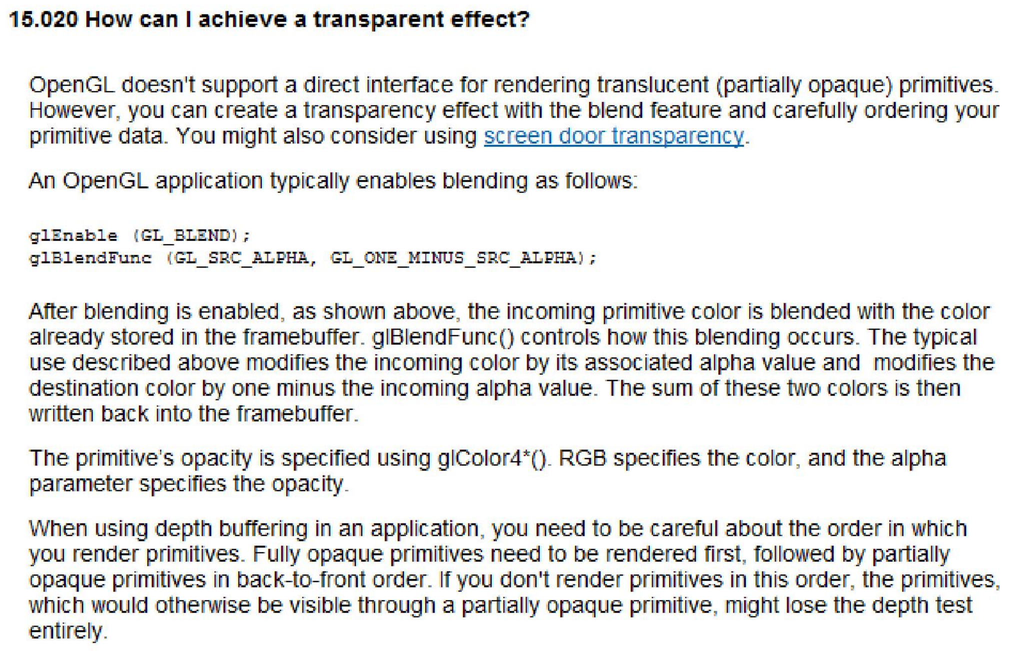
A translucent physical material shows objects behind it, but those objects are obscured by the translucent material. In addition, a translucent material reflects some of the light that hits it, making the material visible. Physical examples of translucent materials include sheer cloth, thin plastic, and smoked glass.

Transparent and translucent are often used synonymously. Materials that are neither transparent nor translucent are opaque.

Blending is OpenGL's mechanism for combining color already in the framebuffer with the color of the incoming primitive. The result of this combination is then stored back in the framebuffer. Blending is frequently used to simulate translucent physical materials. One example is rendering the smoked glass windshield of a car. The driver and interior are still visible, but they are obscured by the dark color of the smoked glass.

http://bit.ly/hRaQgk

Transparency in OpenGL [2]: Blending vs. Screen Door



15.020 How can I achieve a transparent effect?

OpenGL doesn't support a direct interface for rendering translucent (partially opaque) primitives. However, you can create a transparency effect with the blend feature and carefully ordering your primitive data. You might also consider using screen door transparency.

An OpenGL application typically enables blending as follows:

```
glEnable (GL_BLEND);
glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

After blending is enabled, as shown above, the incoming primitive color is blended with the color already stored in the framebuffer. glBlendFunc() controls how this blending occurs. The typical use described above modifies the incoming color by its associated alpha value and modifies the destination color by one minus the incoming alpha value. The sum of these two colors is then written back into the framebuffer.

The primitive's opacity is specified using glColor4*(). RGB specifies the color, and the alpha parameter specifies the opacity.

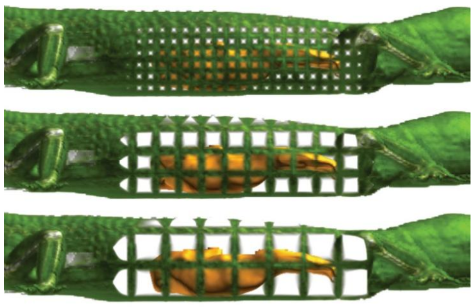
When using depth buffering in an application, you need to be careful about the order in which you render primitives. Fully opaque primitives need to be rendered first, followed by partially opaque primitives in back-to-front order. If you don't render primitives in this order, the primitives, which would otherwise be visible through a partially opaque primitive, might lose the depth test entirely.

http://bit.ly/hRaQgk

Transparency in OpenGL [3]: Screen Door

15.030 How can I create screen door transparency?

This is accomplished by specifying a polygon stipple pattern with glPolygonStipple() and by rendering the transparent primitive with polygon stippling enabled (glEnable(GL_POLYGON_STIPPLE)). The number of bits set in the stipple pattern determine the amount of translucency and opacity; setting more bits result in a more opaque object, and setting fewer bits results in a more translucent object. Screendoor transparency is sometimes preferable to blending, becuase it's order independent (primitives don't need to be rendered in back-to-front order).



Screen door: Viola *et al*. (2004), http://bit.ly/dVEa7I
Technical University of Vienna, IEEE Vis 2004

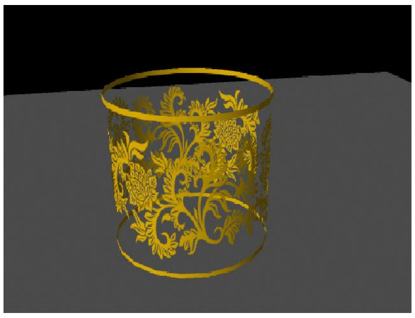
Transparency in OpenGL [4]: Glass

15.040 How can I render glass with OpenGL?

This question is difficult to answer, because what looks like glass to one person might not to another. What follows is a general algorithm to get you started.

First render all opaque objects in your scene. Disable lighting, enable blending, and render your glass geometry with a small alpha value. This should result in a faint rendering of your object in the framebuffer. (Note: You may need to sort your glass geometry, so it's rendered in back to front Z order.)

Now, you need to add the specular highlight. Set your ambient and diffuse material colors to black, and your specular material and light colors to white. Enable lighting. Set glDepthFunc(GL_EQUAL), then render your glass object a second time.



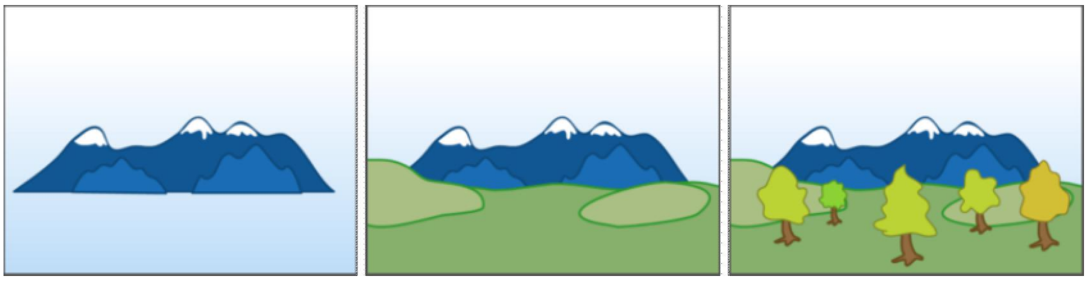
Alpha blending: Lim (2010), http://bit.ly/6TsJrb
Goon Creative, Maya Transparency Tutorial

© 1997 – 2011 Khronos Group
http://bit.ly/hRaQgk

Transparency in OpenGL [5]: Alpha & Painter's Algorithm

15.050 Do I need to render my primitives from back to front for correct rendering of translucent primitives to occur?

If your hardware supports destination alpha, you can experiment with different glBlendFunc() settings that use destination alpha. However, this won't solve all the problems with depth buffered translucent surfaces. The only sure way to achieve visually correct results is to sort and render your primitives from back to front.



© 2004 – 2009 Wikipedia, *Painter's Algorithm*
http://bit.ly/eeebCN

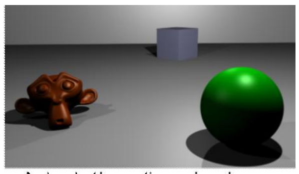
© 1997 – 2011 Khronos Group
http://bit.ly/hRaQgk

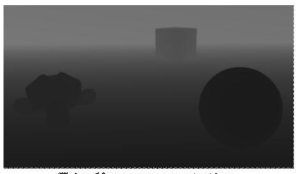
Transparency in OpenGL [6]: Painter's algorithm & Z-buffering

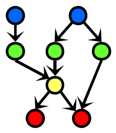
15.070 If I draw a translucent primitive and draw another primitive behind it, I expect the second primitive to show through the first, but it's not there?

Is depth buffering enabled?

If you're drawing a polygon that's behind another polygon, and depth test is enabled, then the new polygon will typically lose the depth test, and no blending will occur. On the other hand, if you've disabled depth test, the new polygon will be blended with the existing polygon, regardless of whether it's behind or in front of it.


A simple three-dimensional scene


Z-buffer representation



Transparency in OpenGL [6]: Partial Transparency

15.080 How can I make part of my texture maps transparent or translucent?

It depends on the effect you're trying to achieve.

If you want blending to occur after the texture has been applied, then use the OpenGL blending feature. Try this:

```
glEnable (GL_BLEND);
glBlendFunc (GL_ONE, GL_ONE);
```

You might want to use the alpha values that result from texture mapping in the blend function. If so, (GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA) is always a good function to start with.

However, if you want blending to occur when the primitive is texture mapped (i.e., you want parts of the texture map to allow the underlying color of the primitive to show through), then don't use OpenGL blending. Instead, you'd use glTexEnv(), and set the texture environment mode to GL_BLEND. In this case, you'd want to leave the texture environment color to its default value of (0,0,0,0).

Summary

- **Reading for Last Class: §4.1 – 4.3, Eberly *2e*; CGA handout**
- **Reading for Today: §2.6, 20.1, Eberly *2e*; OpenGL primer material**
- **Reading for Next Class: §5.1 – 5.2, Eberly *2e***
- **Last Time: Scene Graphs**
  - **Maintaining state**
  - **Coming up: traversal**
- **CGA Demos, Videos**
  - **State of CGA: videos**
  - **Issues: photorealism, hardware, traditional (non-CG) animation**
  - **Techniques showcased: multipass texturing, alpha, portals**
- **Shading and Transparency in OpenGL**
  - **Alpha blending**
  - **Painter's algorithm – less efficient, can handle non-opaque objects**
  - **Depth buffering (*z*-buffering) – in hardware, fast, opaque only**

Terminology

- Non-Photorealistic Rendering
  - Cartoon shaders
  - Pencil shaders
- CGA and Realism
  - Aliasing – reconstructed image differs from original
  - Alias – artifact in reconstructed image (jaggies, Moiré pattern, *etc.*)
  - Anti-aliasing – techniques (*e.g.*, area sampling) for avoiding aliasing
  - Temporal aliasing – aliasing over time (*e.g.*, in animation)
  - Temporal anti-aliasing – smoothing out aliasing over time
- Shading and Transparency in OpenGL
  - Alpha blending – using A channel of R, G, B, A to combine colors
  - Painter's algorithm *aka* priority fill – back-to-front rendering
  - Depth buffering (z-buffering) – checking *z* values