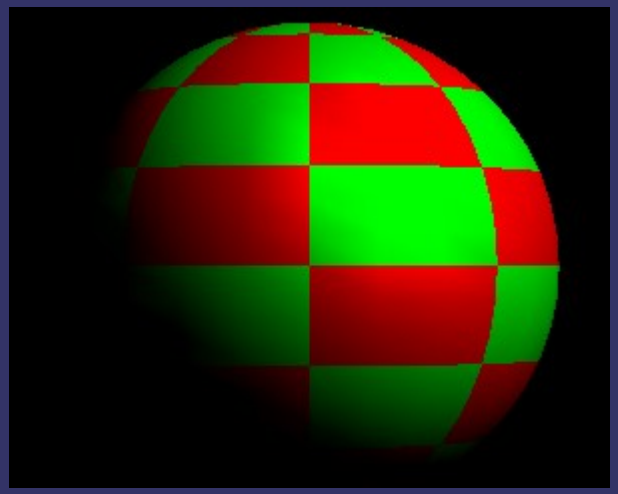
Computer Graphics Programming I

- Agenda:
  - Additive specular reflections
  - Projective textures
  - Point sprites
  - Multi-texture
  - Texture combiners, part 2

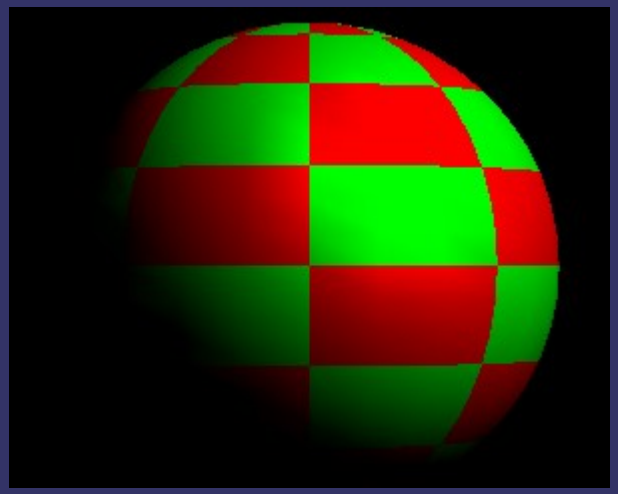
Specular Reflection w/Texture

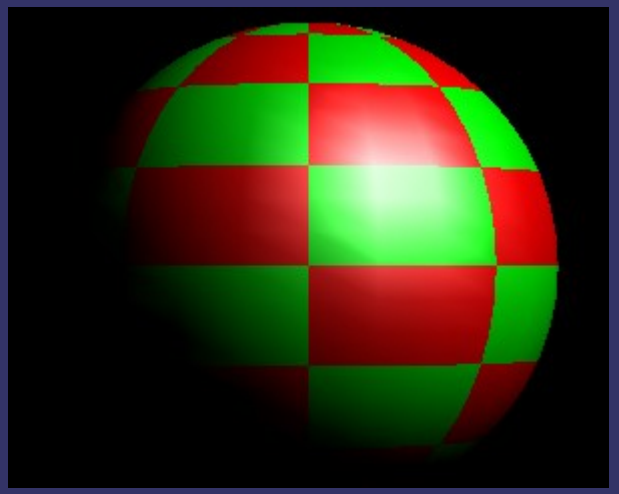
- OpenGL performs lighting and provides a single interpolated color input to the texture combiner.
  - Why is this wrong? (Or at least probably not what is wanted...)



Specular Reflection w/Texture

- OpenGL performs lighting and provides a single interpolated color input to the texture combiner.
  - Why is this wrong? (Or at least probably not what is wanted...)
  - Texture color is typically a diffuse property.





Separate Specular

- Separate specular fixes this.
  - Extension `GL_EXT_separate_specular` is part of core GL 1.2 and later.
  - Provides diffuse color as input to texture combiner.
  - Adds specular color *after* texture application.
- Enable with `glLightModel`:

```
glLightModeli(GL_LIGHT_MODEL_COLOR_CONTROL,
  GL_SEPARATE_SPECULAR_COLOR);
glLightModeli(GL_LIGHT_MODEL_COLOR_CONTROL,
  GL_SINGLE_COLOR);
```

Secondary Color

- Similar functionality without lighting.
- Specify secondary color via `glSecondaryColor3{bsifd ubusui}[v]`.
  - Works just like the various `glColor` calls, but no alpha is specified.
  - Enable the final add by enabling `GL_COLOR_SUM`.
- Extension `GL_EXT_secondary_color` is part of core GL 1.4 and later.
- Important: This gives a little more math that we can do.

Projective Textures

- We can create an effect of a texture being “projected” onto a surface.
  - Like what a movie projector does.
- What makes perspective projection (versus parallel projection) “work”?

Projective Textures

- We can create an effect of a texture being “projected” onto a surface.
  - Like what a movie projector does.
- What makes perspective projection (versus parallel projection) “work”?
  - Dividing by Z.
  - Do the same thing with texture coordinates to get the same effect!
    - Except use the q coordinate.

Usage Overview

- Use `OBJECT_LINEAR` texgen to compute initial texture coordinate as distance from the center of the object.
- Set texture matrix to:
  1) Transform coordinate from object-space to projector-space.
  2) Apply perspective transformation.
  3) Scale & bias from [-1, 1] to [0, 1]
     - Unless you're using a mirrored wrap mode!
- Just like the usual camera transform!

Transform to Projector Space

- Apply usual object-to-world transformations.
- Use `gluLookAt`, for example, to for world-to-projector transformation.
- Use `gluPerspective`, for example, to perform perspective transformation.
  - Do this on texture matrix instead of separate projection matrix!

Scale & Bias

- How do you transform from [-1, 1] to [0, 1]?

Scale & Bias

- How do you transform from [-1, 1] to [0, 1]?
  - Multiply by 0.5 and add 0.5.
- What sort of matrix does that for all coordinates?

Scale & Bias

- How do you transform from [-1, 1] to [0, 1]?
  - Multiply by 0.5 and add 0.5.
- What sort of matrix does that for all coordinates?

$$\begin{bmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Projective Texture Notes

- Can also use `EYE_LINEAR`, but we need the inverse modelview matrix.
  - We'll use this for shadow maps in VGP353.
- The previous operations need to be done in *reverse* order to get the correct matrix.
  - Right?

References

http://developer.nvidia.com/object/Projective_Texture_Mapping.html

Point Sprites

- Having a billboard that always faces the camera can be very useful.
  - Particle effects
- For an arbitrary eye position, how do you calculate the position of a quadrilateral that will face the camera?

Point Sprites

- Having a billboard that always faces the camera can be very useful.
  - Particle effects
- For an arbitrary eye position, how do you calculate the position of a quadrilateral that will face the camera?
  - You probably cry...a lot.
  - It can be done, but it's a waste of CPU time.
- Point sprites do this essentially for free.

Using Point Sprites

- Point sprite mode is enabled with `GL_POINT_SPRITE`.
  - Per-texture unit set `GL_COORD_REPLACE` to `GL_TRUE`.
- Upper left of sprite gets (0, 0, 0, 1) for texture coordinate, and lower right get (1, 1, 0, 1).
  - Yes, this is backwards. *Blame Nvidia.*
  - Or use a texture matrix to “fix” it.
- Each point primitive will behave as usual, but will have these interpolated texture coordinates.

Multi-Texture

- `GL_ARB_multitexture` part of core in 1.3, but almost universally available long before
- Multiple, active textures per-drawing call.
  - Multiple sets of texture coordinates
  - Multiple sets of wrap state
  - Multiple sets of environment state
  - etc.
- Maximum number of texture units queryable:

```
glGetIntegerv(GL_MAX_TEXTURE_UNITS,
  &max_units);
```

Multi-Texture State

- How do we track multiple, independent sets of texture state?
  - Before this extension there was no way to tell any of the operations which texture unit to operate on.

Multi-Texture State

- How do we track multiple, independent sets of texture state?
  - Before this extension there was no way to tell any of the operations which texture unit to operate on.
  - Add the notion of the “active” texture unit.

    ```
    glActiveTexture(GLenum unit);
    ```
  - Modal, like matrix mode.
  - Strange API prevents the need to add new versions of every texture function.
    - Texture unit “parameter” is implied from the global state.

Multiple Texture Coordinates

- Problem: Can't call `glActiveTexture` between begin / end.

Multiple Texture Coordinates

- Problem: Can't call `glActiveTexture` between begin / end.
- Solution: Add `glTexCoord` commands that take the texture unit as a parameter.

```
glMultiTexCoord{234}{bsifd
 ubusui}{v}(GLenum unit, ...);
```

Multiple Texture Combiners

- Combiner state is per-unit.
  - One input is the texel value.
  - The other input is one of:
    - The primary color for unit 0.
    - The output of the previous unit for all other units.

Break

Texture Combiners

- Base OpenGL 1.2 functionality is *very* limited.
- `GL_ARB_texture_env_combine` provides a much better interface.
  - More general
  - More generic
    - Operation performed not dependent on texture format.
  - Extendable
    - Several extensions add more possible operations
  - Became part of core in 1.3

Texture Combine Introduction

- Source for RGB and A of each operand:
  - Primary color, texture color, constant color, or output of previous combiner
- Modifier for RGB and A of each operand:
  - Color, one-minus-color, alpha, or one-minus-alpha.
- Operation for RGB and A:
  - Replace, modulate, add, biased add, interpolate, or subtract.
- Post-scale for RGB and A

Enable Texture Combine

- Enable by setting `GL_COMBINE` as texture environment mode:

```
glTexEnvi(GL_TEXTURE_ENV,
 GL_TEXTURE_ENV_MODE, GL_COMBINE);
```

Set Sources

- Up to 3 sources depending on operation.
  - Named `GL_SOURCE{012}_{RGB,ALPHA}`
- Each source can be one of the following:
  - `GL_TEXTURE` – current texture value
  - `GL_CONSTANT` – per-stage constant color
  - `GL_PRIMARY_COLOR` – interpolated primary color
  - `GL_PREVIOUS` – output of previous combiner stage
- Set via `glTexEnvi`:

```
glTexEnvi(GL_TEXTURE_ENV, GL_SOURCE0_RGB,
GL_TEXTURE);
```

Set Modifiers

- Modifiers named `GL_OPERAND{012}_{RGB,ALPHA}`
- Can be one of:
  - `GL_SRC_COLOR` – value from selected color source
  - `GL_ONE_MINUS_SRC_COLOR` – 1.0 minus value from selected color source
  - `GL_SRC_ALPHA` – value from selected alpha source
  - `GL_ONE_MINUS_SRC_ALPHA` – 1.0 minus value from selected alpha source

Set Modifiers (cont.)

- GL_*_COLOR can only be used with RGB operands
- Also set with glTexEnvi:

```
glTexEnvi(GL_TEXTURE_ENV, GL_OPERANDO_RGB,
  GL_ONE_MINUS_ALPHA);
```

Set Operation

- Six possible operations:
  - `GL_REPLACE` – output is operand 0
  - `GL_MODULATE` / `GL_ADD` / `GL_SUBTRACT` – output is *Arg*0 {+ - * } *Arg*1
  - `GL_ADD_SIGNED` – *Arg*0 + *Arg*1 - 0.5
  - `GL_INTERPOLATE` – *Arg0* * *Arg2* + *Arg1* * (1 – *Arg2*)
- Again, `glTexEnvi` for the win:

```
glTexEnvi(GL_TEXTURE_ENV, GL_COMBINE_RGB,
  GL_SUBTRACT);
```

Set Scale

- Three possible scale factors:
  - 1.0, 2.0, 4.0
- Set with `glTexEnvi` or `glTexEnvf`:

```
glTexEnvi(GL_TEXTURE_ENV, GL_RGB_SCALE, 2);
glTexEnvf(GL_TEXTURE_ENV, GL_ALPHA_SCALE,
  4.0);
```

Dot-product Combiner

- `GL_ARB_texture_env_dot3` part of 1.3 core
- Adds to new combine operations:
  - `GL_DOT3_RGB` – 3 component dot-product of RGB components
  - `GL_DOT3_RGBA` – Like `GL_DOT3_RGB`, but also writes value to alpha component
  - Actual operation pre-biases each component by -0.5, then scales result by 4.0.
    - Usual post-scale is applied after the built-in scale.

Dot-product Combiner (cont.)

- Why pre-bias by -0.5?

Dot-product Combiner (cont.)

- Why pre-bias by -0.5?
  - Range of colors is [0, 1], but components of normals, for example, can be negative.
- Why post-scale by 4.0?

Dot-product Combiner (cont.)

- Why pre-bias by -0.5?
  - Range of colors is [0, 1], but components of normals, for example, can be negative.
- Why post-scale by 4.0?
  - The pre-bias gives a range of [-0.5, 0.5]. Multiplying two values in that range gives a new range or [-0.25, 0.25]. The post-scale expands the range to [-1.0, 1.0].

Related Extensions

- `GL_EXT_texture_env_combine`
  - Like ARB version
  - *Without* `GL_SUBTRACT`.
  - Operands to `GL_INTERPOLATE` more restricted.
  - Lots of older hardware supports this but not ARB version.
- `GL_EXT_texture_env_dot3`
  - Like ARB version, *without* built-in scale by 4.0.
  - AFAIK, only the original Radeon (Radeon 7200) supports this and not the ARB version.

Related Extensions - GL_ARB_texture_env_crossbar

- Part of core since GL 1.4.
- Adds new sources:
  - `GL_TEXTURE<n>` - Use any texture as an input to any stage
- Supported by everyone *except* Nvidia.
  - Has rules about what to do when a disabled unit is referenced that didn't work on Nvidia hardware.
  - That rule was relaxed for GL 1.4.

Related Extensions - GL_ATI_texture_env_combine3

- Adds new operations:
  - `GL_MODULATE_ADD_ATI` – *Arg0 * Arg1 + Arg2*
  - `GL_MODULATE_ADD_SIGNED_ATI` – *Arg0 * Arg1 + Arg2* - 0.5
  - `GL_MODULATE_SUBTRACT_ATI` – *Arg0 * Arg1 – Arg2*
- Adds new sources:
  - `GL_ZERO` and `GL_ONE`
- Supported by all ATI hardware since Radeon 7200.

Related Extensions - GL_NV_texture_env_combine4

- New environment mode (`GL_COMBINE4`) with two operations:
  - `GL_ADD` – (*Arg0* * *Arg1*) + (*Arg2* * *Arg3*)
  - `GL_ADD_SIGNED` – (*Arg0* * *Arg1*) + (*Arg2* * *Arg3*) – 0.5
  - All other modes can be derived from these two!
- Adds new sources:
  - `GL_ZERO` and `GL_TEXTURE<n>`
- Supported on *all* Nvidia hardware since the TNT.

Next week...

- Lighting calculations with texture combiners
  - Tangent space
  - `GL_DOT3_RGB` for the win! :)
- Quiz #3.

Legal Statement

- This work represents the view of the authors and does not necessarily represent the view of IBM or the Art Institute of Portland.
- OpenGL is a trademark of Silicon Graphics, Inc. in the United States, other countries, or both.
- Khronos and OpenGL ES are trademarks of the Khronos Group.
- Other company, product, and service names may be trademarks or service marks of others.