MATERIALS AND TEXTURES
IMPROVING VISUAL FIDELITY
HISTORICALLY GAMES RELIED ON SIMPLE, LOCAL LIGHTING MODELS
HOW DID THEY GET GAMES TO LOOK LIKE THIS?
TEXTURE MAPPING

- Take a “painted” image (texture) and wrap it around a 3D mesh (map) to add more detail
- Works for shading parameters other than color
**BASIC MAPPING**

- Textures live in a 2D space
  - Parameterize points in the texture with 2 coordinates: \((s, t)\)
- Define the mapping from \((x, y, z)\) in world space to \((s, t)\) in texture space
- For polygons:
  - Specify \((s, t)\) coordinates at vertices
  - Interpolate \((s, t)\) for other points
BASIC TEXTURING CONCEPTS AND TERMS

- Vertices are coordinates that define geometry
- Texture coordinates specified at vertices and interpolated across triangles
- Texture values for points mapping outside the texture image can be generated in various ways:
  - REPEAT, CLAMP, etc
- Width and height of texture images is often constrained
  - Powers of two
  - Sometimes required to be a square
**MULTITEXTURING**

- Multitexturing hardware provides pipeline of texture units to work on fragments
  - Apply multiple textures to a primitive in the same pass
- Example
  1. Apply color map
  2. Modify illumination to simulate bumps
  3. Modify opacity
- Note that this is not the same as a multi-pass rendering – done in a single pass!
Modern game engines provide a lot of texture support

- Automatic texture atlasing
- High level of control for importing and editing
- Varying resolutions to support hardware performance

Artist tools often included for texture management

- Design texture images
- Specify how to apply to object
- Profiling to maintain good memory and performance bounds
**TEXTURE ATLASING**

- A packed set of textures (or sprites)
- 2D Example: a sprite sheet
TEXTURE ATLASING

- Also used in 3D games!
- Artists pack the textures for many objects into one image
  - The texture coordinates for a given object may only index into a small part of the image
  - Care must be taken at sub-image boundary to achieve correct blending
- Mipmapmapping is restricted
- Best for objects that are at a known resolution
COMBINING TEXTURES
MIPMAPS

- Store multiple resolutions of same texture
- Sample based on distance from camera
OTHER TEXTURE ISSUES

TEXTURE SAMPLING (ALIASING) CREATES VISUAL ARTIFACTS

https://www.iquilezles.org/www/articles/filtering/filtering.htm
A LARGER EXAMPLE
3D EXAMPLE: TEXTURE TOOL
OTHER TEXTURING TECHNIQUES

- Animated textures
  - Texture matrix transforms texture in memory
  - Texture can slide, rotate, and stretch/shrink over surface
  - Useful for things like flame, swirling vortices, or pulsing entrances...

- Projective textures
  - Texture projected onto the scene as if from a slide projector
  - Used in light maps, shadow maps and decals
SHADOW MAPS

- Render shadows by determining if pixels are occluded from light sources
  
  1. Render scene from light source’s point of view (multiple renders for multiple light sources)
  
  2. Store depth values of this scene as a texture (the shadow map)
  
  3. Render scene from camera’s point of view and test if object coordinates are lit or unlit by light

- Must transform objects in scene into light source’s coordinate system

- Check depth of object against depth of shadow map value to determine if object is occluded
SHADOW MAP EXAMPLE
SHADOW MAP CHALLENGES

- Basic shadow mapping only generates hard shadows
  - Need additional processing for shadow penumbra
- Resolution of shadow map determines resolution of shadow
  - Aliasing and continuity issues
- Resource intensive
  - Need pre-baking or advanced techniques
**MSAA AND TEMPORAL ANTI-ALIASING**

- Multisample anti-aliasing is a form of supersampling (oversampling to reduce loss of the signal)
  - Naive oversampling samples the entire image at higher resolution then reduces
  - Observation: aliasing occurs in specific areas rather than universally
  - Solution: only perform super sampling in areas with discontinuities in triangles/depth/etc

- Temporal anti-aliasing samples pixel over time to reduce temporal aliasing
  - Temporal aliasing occurs when objects move faster than frame speed
  - Apply filters based on multiple frames to soften effect
Environment mapping produces reflections on shiny objects.

Texture is transferred in the direction of the reflected ray from the environment map onto the object.
ENVIRONMENT MAPPING CONT’D

- Reflected ray: \( R = I - 2(N \cdot I)N \)
EXAMPLE
CUBE MAPPING

- The map resides on the surfaces of a cube around the object
  - Typically align the faces of the cube with the coordinate axes
- Can make map rendering arbitrarily complex as its possible to do off-line
- For each face of the cube either:
  - Render the world from the center of the object with the cube face as the image plane
  - Or take 6 photos of a real environment with a camera in the object’s position
CUBE MAP EXAMPLE
WHAT DO TEXTURES REPRESENT?

- The graphics hardware doesn’t know what is in a texture.
- It applies a set of operations using values it finds in the texture, the existing value of the fragment (pixel), and maybe another color.
- The programmer decides what these operations are.
- Examples:
  - Scalar luminance data (multiplies the fragment color).
  - Alpha data (multiplies the fragment’s alpha channel).
  - Vector data (modifies the surface normals).
  - Depth data (determines distance from light source for shadow mapping).
TEXTURES IN DIGITAL ART

- Assets designed for modern graphics pipeline
  - Low-poly, high-texture
  - Multiple maps for multiple effects
- Example:
  - https://www.artstation.com/artwork/2yVKB
  (Arnab Roy, Maya)
PHYSCALLY-BASED MATERIALS

- Textures provide more details but Phong lighting model inherently limited
  - Function is very approximate and not physically-based
- Can improve material model by using functions based on the physics of light
BRDFs

- Bidirectional reflectance distribution function
- Defines how a material reflects light based on the angle of observation
- Determines ratio of reflected radiance
  - Physically-based
  - Empirically studied by material sample
THE RENDERING EQUATION

- Describes radiance of light entering and leaving a point

\[ L_o(x, \omega_o, \lambda, t) = L_e(x, \omega_o, \lambda, t) + \int_{\Omega} f_r(x, \omega_i, \omega_o, \lambda, t) L_i(x, \omega_i, \lambda, t) (\omega_i \cdot n) \, d\omega_i \]
ADDITIONAL FUNCTIONS

- **BTDF (bidirectional transmittance distribution function)** models the scattering of transmitted light
- **BSSRDF (bidirectional scattering-surface reflectance distribution function)** model subsurface scattering and related effects
- **BSDF (bidirectional scattering distribution function)** encompasses BRDFs, BTDFs, and BSSRDFs
MATERIAL PARAMETERIZATION

- Base Color (Albedo)
  - Diffuse color based on scattering/absorption of light wavelengths
- Roughness
  - Amount of microsurfaces and imperfections on material’s surface leading to light scatter
- Metallic
  - Degree of “metalness” including colored reflections and any diffusion from corrosion/dirt on surface
- Reflectance
  - Amount of reflected light on non-metallic surfaces
ALBEDO

Reflection

Diffusion
ROUGHNESS

“Blurry” Reflection
METALLIC

(a) Reflected light

Incident light

(b) Metal

![Graph showing reflectance vs. wavelength for different metals](image)
REFLECTANCE

- Chrome
- Rubber

Reflectivity

"Center" to "Edge"
THE ORDER: 1886
MATERIAL TEMPLATES

- Store parameterization in base material
  - Changes from base material store on derived material
  - Global changes to base material change all derived material
- Material templates include:
  - Glasses
  - Masonry
  - Metals
  - Wood
  - etc
- Material compositing done using reference material and blend mask
MATERIAL TEMPLATING
MATERIAL PIPELINE

- BRDFs provide a way for artists to interact with photorealistic lighting models and shader programming at a higher level

- Substance demo reel
  - https://www.youtube.com/watch?v=BYQpPK-qrTM

- Substance overview:
  - https://www.youtube.com/watch?v=ScttSShgXlw
GAMES THAT USE PBR MATERIALS...

- Industry standard so pretty much everyone...
YES, I MEAN EVERYONE

- Non-photorealistic rendering also benefits from PBR models!
**Shader Code**

- Allows (relatively) easy writing of code to transform vertices, geometry, and pixels on the GPU
  - GLSL is language for OpenGL
  - HLSL is language for DirectX
- Setup for sending data to GPU done by graphics library
  - e.g. vertices to process, textures, lights, etc
- Programs on GPU run in parallel for every vertex, shape, pixel, etc being processed
SHADERS IN GODOT

- Godot provides its own language based on GLSL ES 3.0
  - Adds functionality
  - Reduces flexibility
  - Only supports vertex and fragment shaders
- Easier to set up and avoids low level issues
- Assumes some knowledge of shader programming in GLSL
Modern game engines and material programs provide interfaces for artists to work with shaders using visual scripting.
REFERENCES/RESOURCES

- [http://developer.download.nvidia.com/CgTutorial/cg_tutorial_chapter07.html]
- [https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping]
- [https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch08.html]
- [https://marmoset.co/posts/basic-theory-of-physically-based-rendering/]
REFERENCES/RESOURCES

- [https://eng.libretexts.org/Bookshelves/Materials_Science/Supplemental_Modules_(Materials_Science)/Optical_Properties/Metallic_Reflection]
- [https://godotengine.org/article/visual-shader-editor-back]
- [https://www.gdcvault.com/play/1020162/Crafting-a-Next-Gen-Material]