**Ray Tracing**

* Rendering- three-dimensional objects: taking a scene, or model, composed of many geometric objects arranged in 3D space and producing a 2D image that shows the objects as viewed from a particular viewpoint
* Object-order rendering - each object is considered in turn, and for each object all the pixels that it influences are found and updated, the “for each object” loop is on the outside
* Image-order rendering- each pixel is considered in turn, and for each pixel all the objects that influence it are found and the pixel value is computed, the “for each pixel” loop is on the outside
* Render-to-texture: allows you to create texture maps based on an object's appearance in the rendered scene, these would be hard to do if we didn’t have render-to-texture (Shadow mapping, Reflection mapping)
* procedural texture is a texture created using a mathematical description (i.e. an algorithm) rather than directly stored data. The advantage of this approach is low storage cost, unlimited texture resolution and easy texture mapping
* The vertex shader gives information on the three vertices of each triangle. This means if you do a texture lookup here, it will only give you three colors from the texture, thus giving you a triangle that is colored with the gradient of three colors. For example, if you have a complex texture like a hedge with a large triangle like that on the side of a cube we've done in class and in projects, this would just give you a disgusting gradient combination of three shades of green, whereas if you did it in the fragment shader you'd get something that actually looks like the hedge on the side of your cube. However, if your 3d model is detailed enough, that is if you have so many triangles that each triangle is essentially a pixel, performing the texture lookup in the vertex shader would essentially be the same as the fragment shader.
* Multi-texture includes bump map and decal and environment mapping
* Render-to-texture includes reflection and shadow mapping

**The Basic Ray-Tracing Algorithm**

* + Ray tracer works by computing one pixel at a time, and for each pixel the basic task is to find the object that is seen at that pixel’s position in the image
  + A basic ray tracer therefore has three parts:
    - *ray generation*, which computes the origin and direction of each pixel’s viewing ray based on the camera geometry;
    - *ray intersection*, which finds the closest object intersecting the viewing ray;
    - *shading*, which computes the pixel color based on the results of ray inter- section.

**Shading**

* + The important variables in light reflection are the light direction l, which is a unit vector pointing towards the light source; the view direction v, which is a unit vector pointing toward the eye or camera; the surface normal n, which is a unit vector perpendicular to the surface at the point where reflection is taking place
  + Lambertian shading (diffuse): L = kd I max(0, n · l), where L is the pixel color; kd is the *diffuse coefficient*, or the surface color; and I is the intensity of the light source, it is view independent
  + Blinn-Phong shading (specular): h = (v+l) / ∥v + l∥, L = ks I max(0, n · h)^p, where ks is the specular coefficient, or the specular color, of the surface and p controls the focus of light (larger the more focused)
  + Ambient shading: L = ka \* Ia, where ka is the surface’s ambient coefficient, or “ambient color,” and Ia is the ambient light intensity
  + Superposition- the effect caused by more than one light source is simply the sum of the effects of the light sources individually
  + Ideal Specular Reflection/Mirror Reflection: some energy is lost when the light reflects from the surface, and this loss can be different for different colors

**Rasterization**

* The central operation in object-order graphics, and the *rasterizer* is central to any graphics pipeline
* Rasterizer has two jobs: it enumerates the pixels that are covered by the primitive and it interpolates values, called attributes, across the primitive, the output of the rasterizer is a set of fragments, one for each pixel covered by the primitive
* *clipping* operation that removes parts of primitives that could extend behind the eye

**Operations Before and After Rasterization**

* *vertex- processing:* incoming vertices are transformed by the modeling, viewing, and projection transformations, mapping them from their original coordinates into screen space
* After rasterization, further processing is done to compute a color and depth for each fragment
* Per-Vertex Shading: the disadvantage that it cannot produce any details in the shading that are smaller than the primitives used to draw the surface, because it only computes shading once for each vertex and never in between vertices, provides normal vectors at the vertices, and the positions and colors of the lights are provided separately (they don’t vary across the surface, so they don’t need to be specified for each vertex). For each vertex, the direction to the viewer and the direction to each light are computed based on the positions of the camera, the lights, and the vertex. The desired shading equation is evaluated to compute a color, which is then passed to the rasterizer as the vertex color
* Per-fragment Shading: the same shading equations are evaluated, but they are evaluated for each fragment using interpolated vectors, rather than for each vertex using the vectors from the application,
* antialiasing is a software technique for diminishing jaggies - stairstep-like lines that should be smooth. Jaggies occur because the output device, the monitor or printer, doesn't have a high enough resolution to represent a smooth line. Antialiasing reduces the prominence of jaggies by surrounding the stairsteps with intermediate shades of gray

**Bump Textures**

* change the surface normal to give an illusion of fine-scale geometry on the sur- face, negatives are that bumps neither cast shadows nor affect the silhouette of the object

**Displacement Mapping**

* changes the geometry using a texture, common simplification is that the displacement will be in the direction of the surface normal

**Environment Maps/Cube Map**

* can be implemented as a background function that takes in a viewing direction b and returns a RGB color from a texture map, an infinitely large cube with a texture on each face

**Shadow Maps**

* if we rendered the scene using the location of a light source as the eye, the visible surfaces would all be lit, and the hidden surfaces would all be in shadow

**Magnification Aliasing**

* occurs when we have too few texture pixels, which are mapped to many more display pixels
* Nearest Neighbor Reconstruction: sets the color of a display pixel to the texture pixel that maps closest to it, exhibits blocky stairstep artifacts
* Bilinear Reconstruction: determines the color of a texture sample located between texture image pixel locations by weighting the colors of texture image pixels, blurs the boundary between checkerboard squares

**Minification Aliasing**

* happens when we have more than enough texture pixels that map to too few display pixels
* The MIP map provides an estimate of an average texture pixel value across a region of pixels

Tangent is found by taking derivative and if dot product is 0 then it is perpendicular