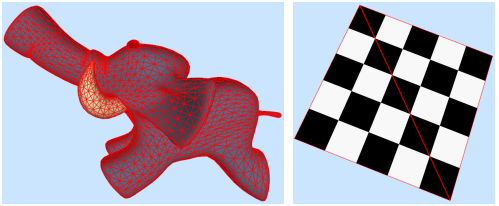
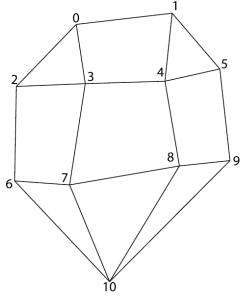
Triangle meshes



- The *indexed face set* is a popular data representation of polygon meshes.
- Any polygon mesh can be converted to a triangle mesh.



VERTICES
0: (-0.2, 1.5, 0)
1: (1.3, 1.7, 0)
2: (-1.1, 0.4, 0)
3: (0.0, 0.45, 1)
4: (1.1, 0.5, 1.2)
5: (2.1, 0.75, 0.2)
6: (-1.2, -1,0.01)
7: (-0.3, -1.2,2)
8: (1.3,-0.9, 3)
9: (2.0 -0.8,1.2)
10: (0.4, -2.1, -1.1)

FACES
0: 0,2,3
1: 0,3,4,1
2: 1,4,5
3: 2,6,7,3
4: 3,7,8,4
5: 4,8,9,5
6: 6,10,7
7: 7,10,8
8: 8,10,9

Figure from Bærentzen et al. *Guide to Computational Geometry Processing*, Springer, 2012.

The main loop in ray tracing

```
for (each pixel row y)                    // This loop is in the render function of RenderEngine
  for (each pixel column x) {             // Implement this loop (first assignment)
    // In the compute_pixel function of RayCaster
    r = ray through screen space position (x + 0.5, y + 0.5);    // Get from camera
    r.tmax = RT_DEFAULT_MAX;
    // In the closest_hit function of Accelerator
    for (each object in scene)
      if (object is intersected) {
        record object material and intersection information;    // In object intersect functions
        r.tmax = distance to intersection;
      }
    // In the compute_pixel function of the ray caster
    if (an object was hit) {
      // In the shade function of Lambertian
      result = 0;
      for (each light source)
        result += light scattered from source to camera at the intersection point;
    }
    else result = background colour;
    store result in pixel (x, y) of the image array;    // In the render function (first assignment)
  }
```

Shading pixels (local illumination)

```
// Starting in Lambertian.cpp with a ray that hit a surface
for (each light source) {
   construct a variable for accumulating light from this source;
   for (each light source sample) {
      // Handle the following three lines by calling the function sample(...)
      // associated with the light source.
      if (ray from surface position to sample point is not occluded) {
         get the direction toward the sample point on the light;
         compute the amount of radiance incident from the sample point;
         if (cosine of angle between surface normal and direction is positive)
            accumulate incident light multiplied by cosine term;
      }
   }
   add accumulated light divided by number of samples to final result;
}
multiply final result by diffuse reflectance and add emission;
```

Lambert's cosine law

brightness decreases in the same ratio by which the sine of the angle of incidence decreases [Lambert 1760]

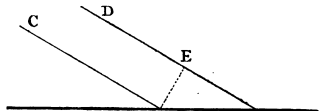


Fig. 1.

- Lambert uses the angle ($\mathrm{CAF} = \mathrm{DBF}$) between the direction of the rays (CA and DB) and the surface tangent plane (AB) as the angle of incidence.
- If we instead measure the angle of incidence θ from the normalised surface normal $\vec{n}$ to the direction toward the incident light $\vec{\omega}'$, sine becomes cosine.
- Then the diffusely reflected light is

$$L_r = \frac{\rho_d}{\pi} L_i \cos\theta = \frac{\rho_d}{\pi} L_i \left(\vec{n} \cdot \vec{\omega}'\right) \ .$$

where ρ_d is the diffuse reflectance.