## 2.4 Rasterization

Given the transformed and projected vertices with their associated shading data (all from geometry processing), the goal of the next stage is to find all pixels—short for *picture elements*—that are inside the primitive, e.g., a triangle, being rendered. We call this process *rasterization*, and it is split up into two functional substages: triangle setup (also called primitive assembly) and triangle traversal. These are shown to the left in Figure 2.8. Note that these can handle points and lines as well, but since triangles are most common, the substages have "triangle" in their names. Rasterization, also called *scan conversion*, is thus the conversion from two-dimensional vertices in screen space—each with a $z$-value (depth value) and various shading information associated with each vertex—into pixels on the screen. Rasterization can also be thought of as a synchronization point between geometry processing and pixel processing, since it is here that triangles are formed from three vertices and eventually sent down to pixel processing.

Whether the triangle is considered to overlap the pixel depends on how you have set up the GPU's pipeline. For example, you may use point sampling to determine "insideness." The simplest case uses a single point sample in the center of each pixel, and so if that center point is inside the triangle then the corresponding pixel is considered inside the triangle as well. You may also use more than one sample per pixel using supersampling or multisampling antialiasing techniques (Section 5.4.2). Yet another way is to use conservative rasterization, where the definition is that a pixel is "inside" the triangle if at least part of the pixel overlaps with the triangle (Section 23.1.2).

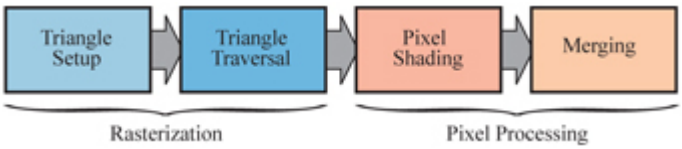


**Figure 2.8** Left: rasterization split into two functional stages, called triangle setup and triangle traversal. Right: pixel processing split into two functional stages, namely, pixel processing and merging.

### 2.4.1 Triangle Setup

In this stage the differentials, edge equations, and other data for the triangle are computed. These data may be used for triangle traversal (Section 2.4.2), as well as for interpolation of the various shading data produced by the geometry stage. Fixedfunction hardware is used for this task.

### 2.4.2 Triangle Traversal

Here is where each pixel that has its center (or a sample) covered by the triangle is checked and a *fragment* generated for the part of the pixel that overlaps the triangle. More elaborate sampling methods can be found in Section 5.4. Finding which samples or pixels are inside a triangle is often called *triangle traversal*. Each triangle fragment's properties are generated using data interpolated among the three triangle vertices (Chapter 5). These properties include the fragment's depth, as well as any shading data from the geometry stage. McCormack et al. [1162] offer more information on triangle traversal. It is also here that perspective-correct interpolation over the triangles is performed [694] (Section 23.1.1). All pixels or samples that are inside a primitive are then sent to the pixel processing stage, described next.