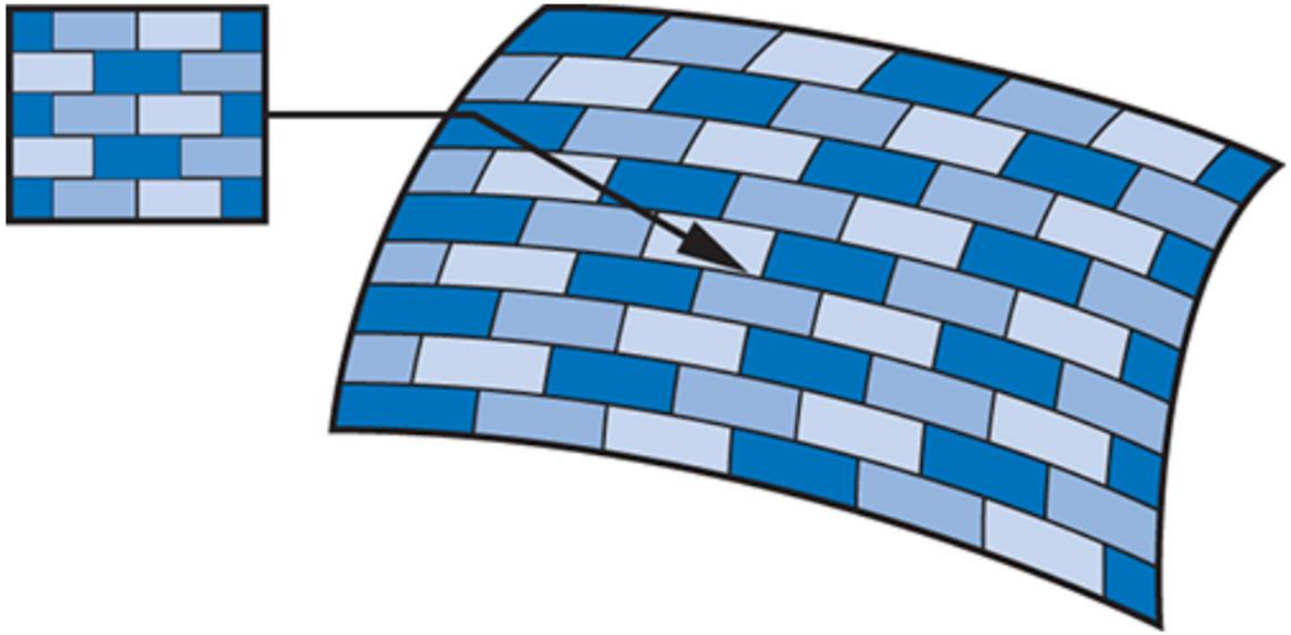
COSC 414/519I: Computer Graphics

2023W2

Shan Du

Texture Mapping

- Mapping texture from a single two-dimensional image to a geometric object.



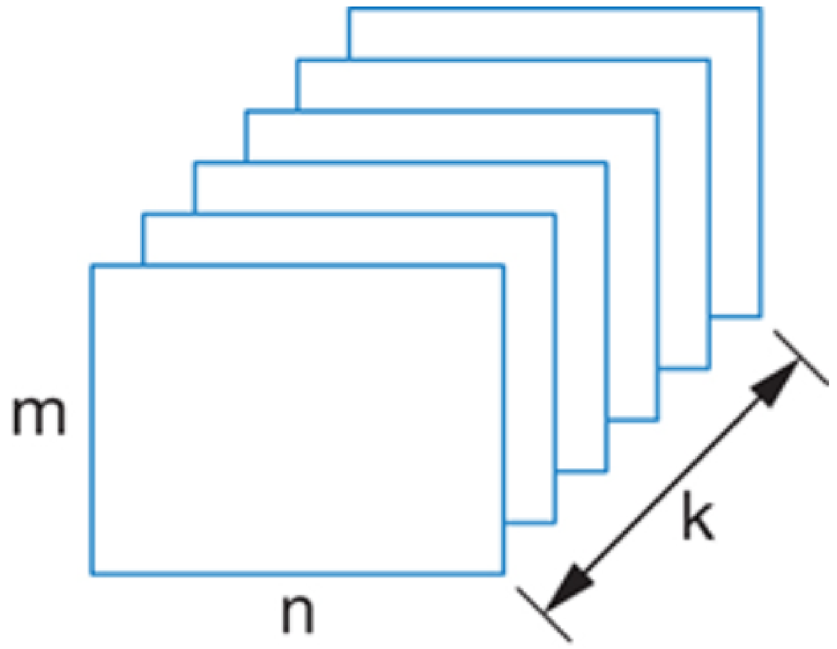
Texture Mapping

- The role of the texture mapping process is to assign the texture image’s pixel colors to the fragments generated by the rasterization process.

Buffer

- Two types of standard buffers: color buffers and depth buffers.
- Inherently discrete: they have limited resolution, both spatially and in depth.

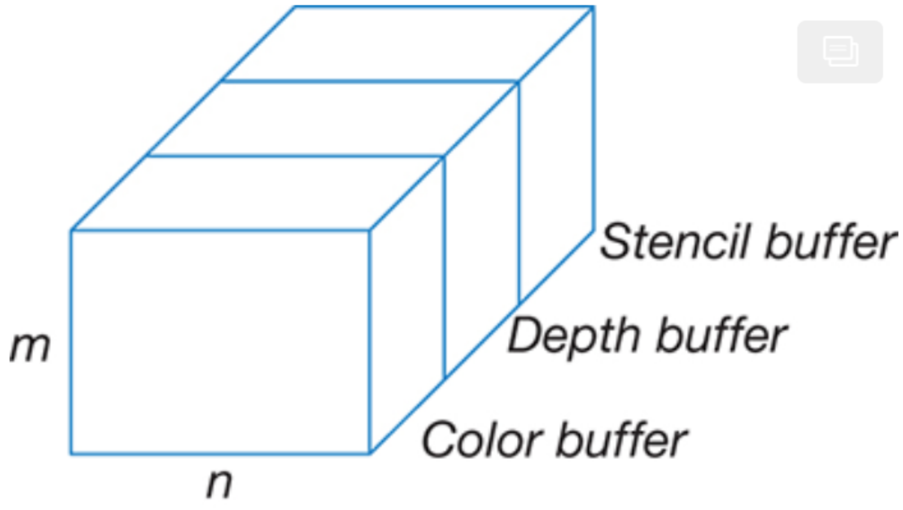
- We can define a (two-dimensional) **buffer** as a block of memory with *n*×*m* *k*-bit elements.



Buffer

- We have used the term *framebuffer* to mean the set of buffers that the graphics system uses for rendering, including the color buffer, the depth buffer, and other buffers the hardware may provide. These buffers generally reside on the graphics card.

Figure 7.2 WebGL framebuffer.



Buffer

- At a given spatial location in the framebuffer, the *k* bits will be partitioned for storing a collection of values for color, depth, and stencil masks. Various data types, such as 16- and 32-bit floating-point values, 16- and 32-bit integers, and potentially even fixed-point values will be used to represent the values for the respective buffers.

Buffer

- We use the term *bitplane* to refer to any of the k $n \times m$ planes in a buffer, and pixel to refer to all k of the bits at a particular spatial location. With this definition, a pixel can be a byte, an integer, or even a floating-point number, depending on which buffer is used and how data are stored in the buffer.

Mapping Methods

- Surface rendering
  - Modelled shape and color still lack the fine surface detail of the real object (e.g., orange).
  - To add this detail by adding more polygons to our model can overwhelm the pipeline.
  - An alternative is not to attempt to build increasingly more complex models, but rather to build a simple model and to add detail as part of the rendering process.

Mapping Methods

- Fragments carry color, depth, and other information that can be used to determine how they contribute to the pixels to which they correspond.
- We use the modified Phong model to determine vertex colors that could be interpolated across surfaces. However, these colors can be modified during fragment processing after rasterization.

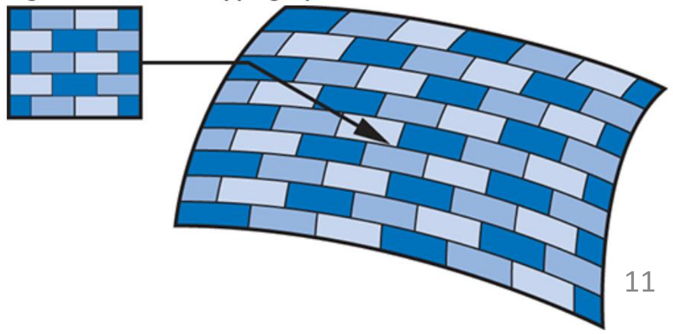
Mapping Methods

- The mapping algorithms can be thought of as either modifying the shading algorithm based on a two-dimensional array— the map—or modifying the shading by using the map to alter the surface using three major techniques:
  - Texture mapping
  - Bump mapping
  - Environment mapping

Mapping Methods

- **Texture mapping** uses an image (or texture) to influence the color of a fragment. Textures can be specified using a fixed pattern, such as the regular patterns often used to fill polygons; by a procedural texture-generation method; or through a digitized image. In all cases, we can characterize the resulting image as the mapping of a texture to a surface, which is carried out as part of the rendering of the surface.

Figure 7.4 Texture mapping a pattern to a surface.



Mapping Methods

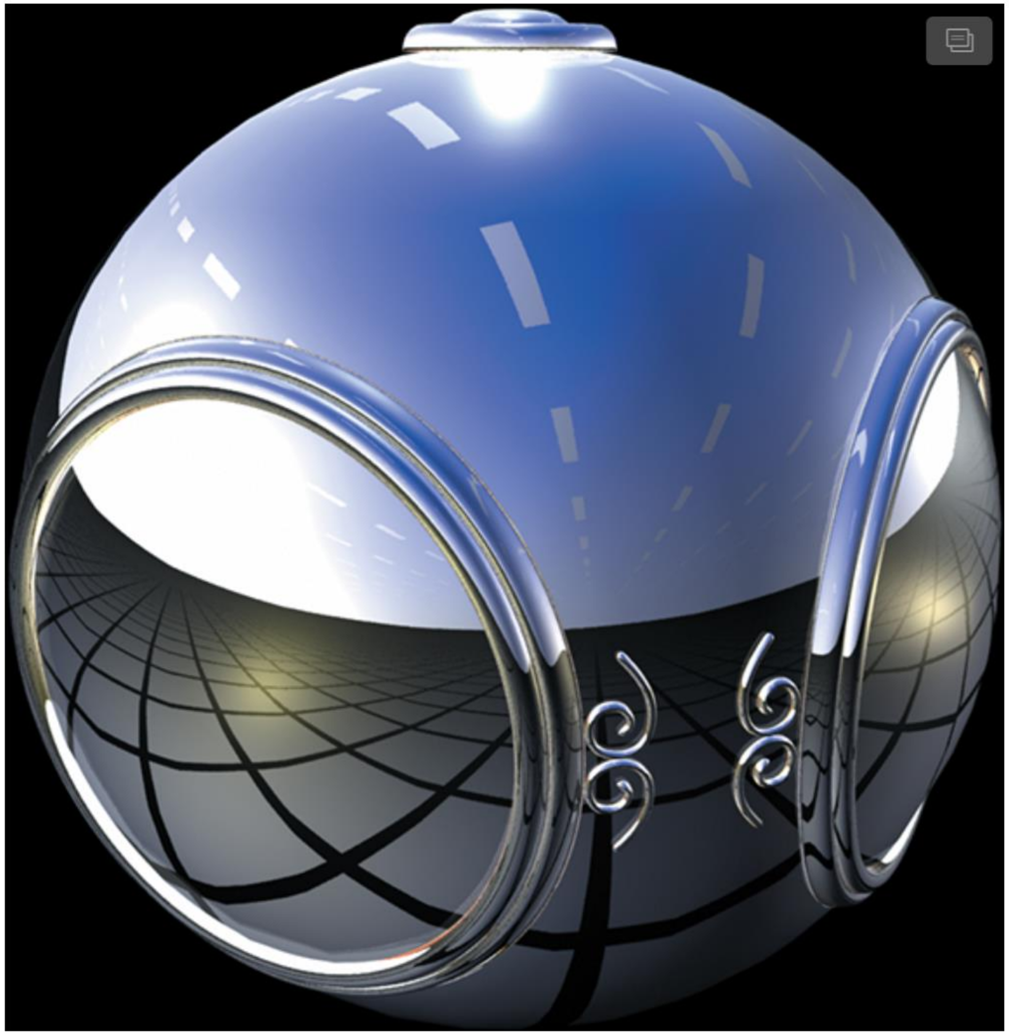
- Whereas **texture maps** give detail by painting patterns onto smooth surfaces, **bump maps** distort the normal vectors during the shading process to make the surface appear to have small variations in shape, such as the bumps on a real orange.
- **Reflection maps**, or **environment maps**, allow us to create images that have the appearance of reflected materials without having to trace reflected rays. In this technique, an image of the environment is painted onto the surface as that surface is being rendered.

Mapping Methods

- The three methods have much in common. All three alter the shading of individual fragments as part of fragment processing. All rely on the map being stored as a one-, two-, or three-dimensional digital image. All keep the geometric complexity low while creating the illusion of complex geometry. However, all are also subject to aliasing errors.

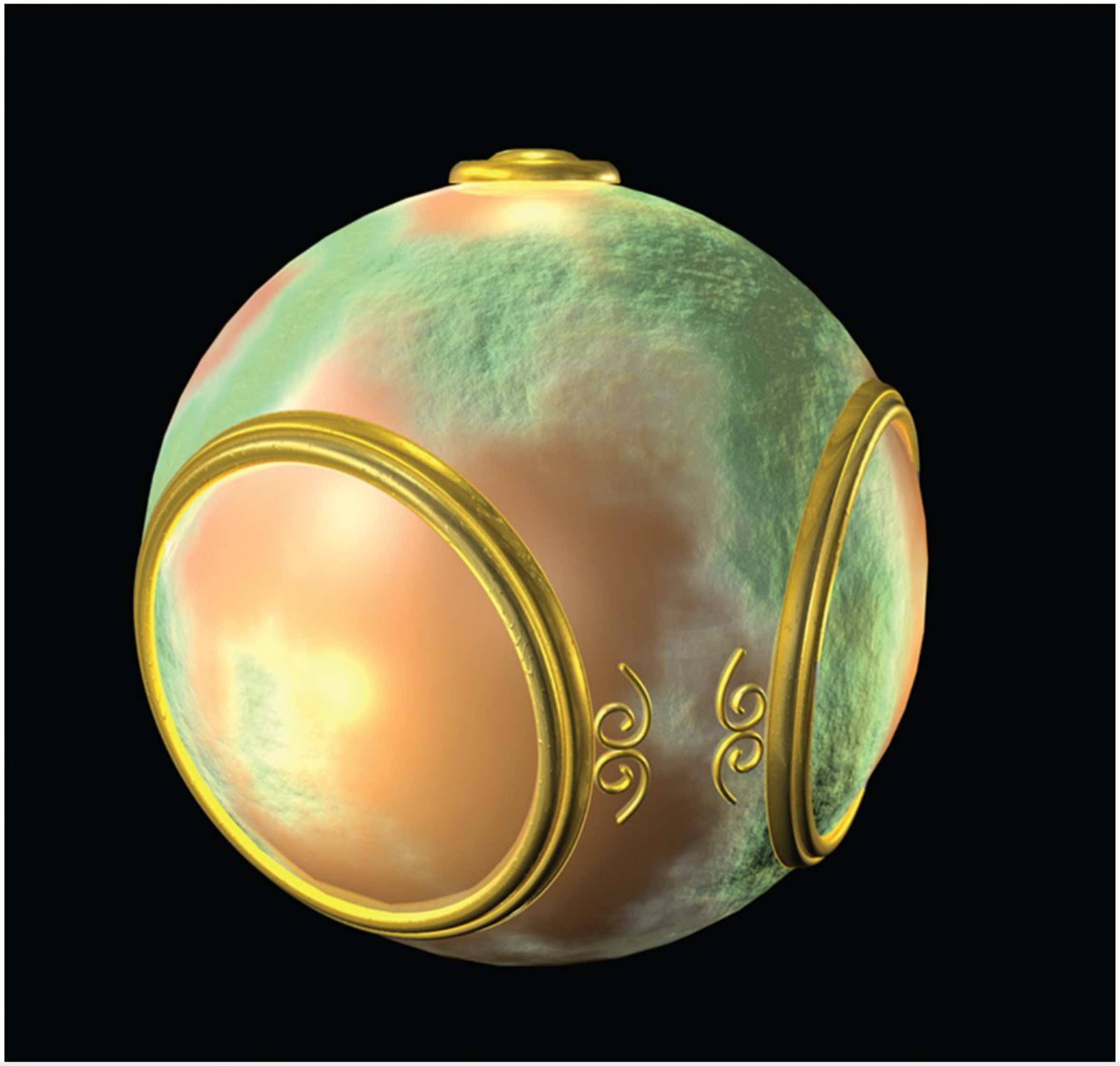
Mapping Methods

Figure 1.40 Rendering of sun object using an environment map.



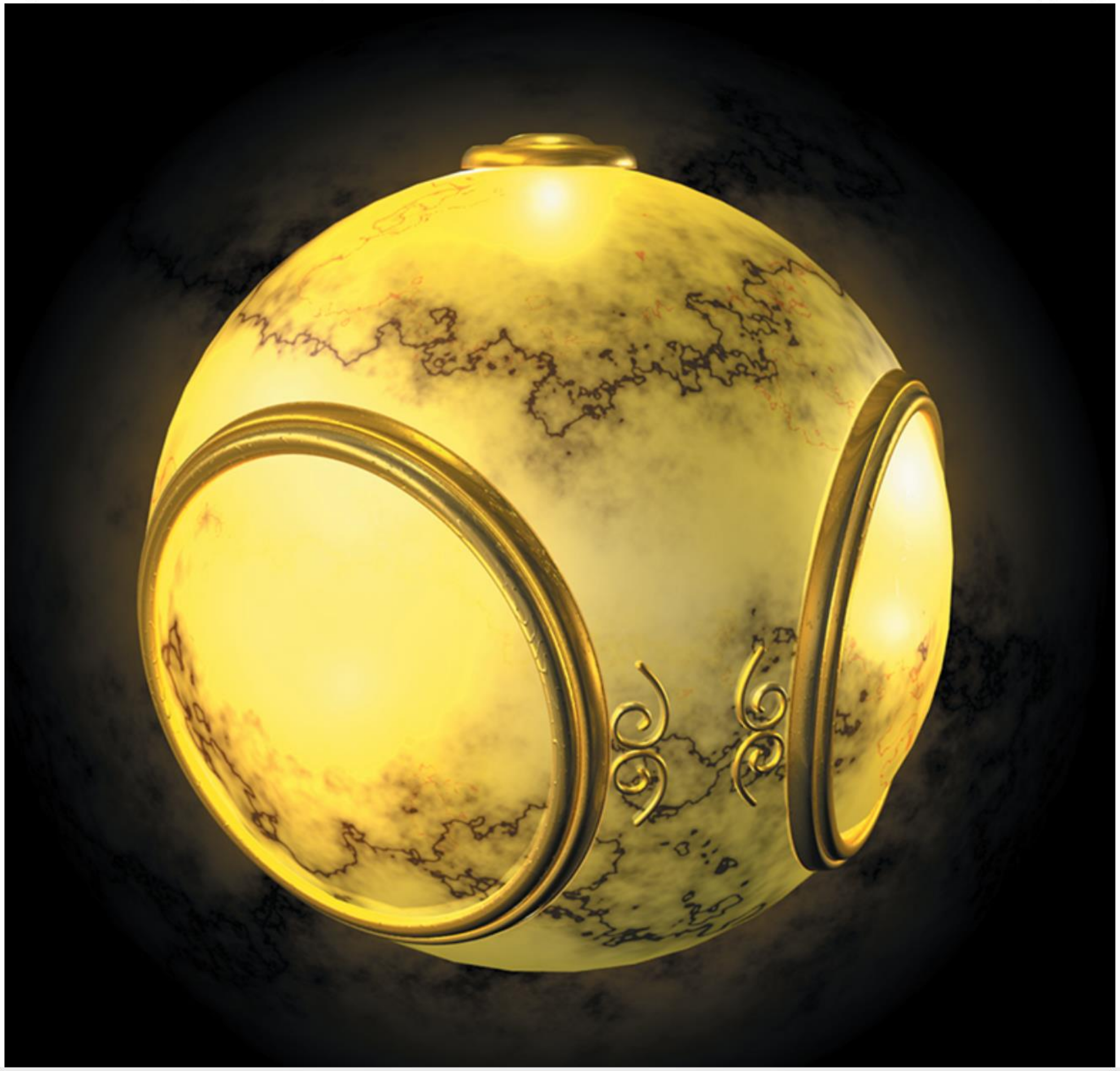
Mapping Methods

Figure 1.39 Rendering of sun object using a bump map.



Mapping Methods

Figure 1.34 Image of sun object created with NURBS surfaces and rendered with texture mapping.



Mapping Methods

- In virtual reality, visualization simulations, and interactive games, real-time performance is required. Hardware support for texture mapping in modern systems allows the detail to be added without significantly degrading the rendering time.

2D Texture Mapping

- Textures are patterns. They can range from regular patterns, such as stripes and checkerboards, to the complex patterns that characterize natural materials. In the real world, we can distinguish among objects of similar size and shape by their textures. If we want to create more detailed virtual images, we can extend our present capabilities by mapping a texture to the objects that we create.

2D Texture Mapping

- Textures can be one-, two-, three-, or four-dimensional. For example, a one-dimensional texture might be used to create a pattern for coloring a curve. A three-dimensional texture might describe a solid block of material from which we could sculpt an object.
- Because the use of surfaces is so important in computer graphics, mapping two-dimensional textures to surfaces is by far the most common use of texture mapping.
- However, the processes by which we map these entities is much the same regardless of the dimensionality of the texture, and we lose little by concentrating on two-dimensional texture mapping.

2D Texture Mapping

- Although there are multiple approaches to texture mapping, all require a sequence of steps that involve mappings among three or four different coordinate systems. At various stages in the process, we will be working with **screen coordinates**, where the final image is produced; **object coordinates**, where we describe the objects upon which the textures will be mapped; **texture coordinates**, which we use to locate positions in the texture; and **parametric coordinates**, which we use to specify parametric surfaces.

2D Texture Mapping

- Textures start out as two-dimensional images. Thus, they might be formed by application programs or scanned in from a photograph, but, regardless of their origin, they are eventually brought into processor memory as arrays. We call the elements of these arrays **texels**, or texture elements, rather than pixels to emphasize how they will be used.

2D Texture Mapping

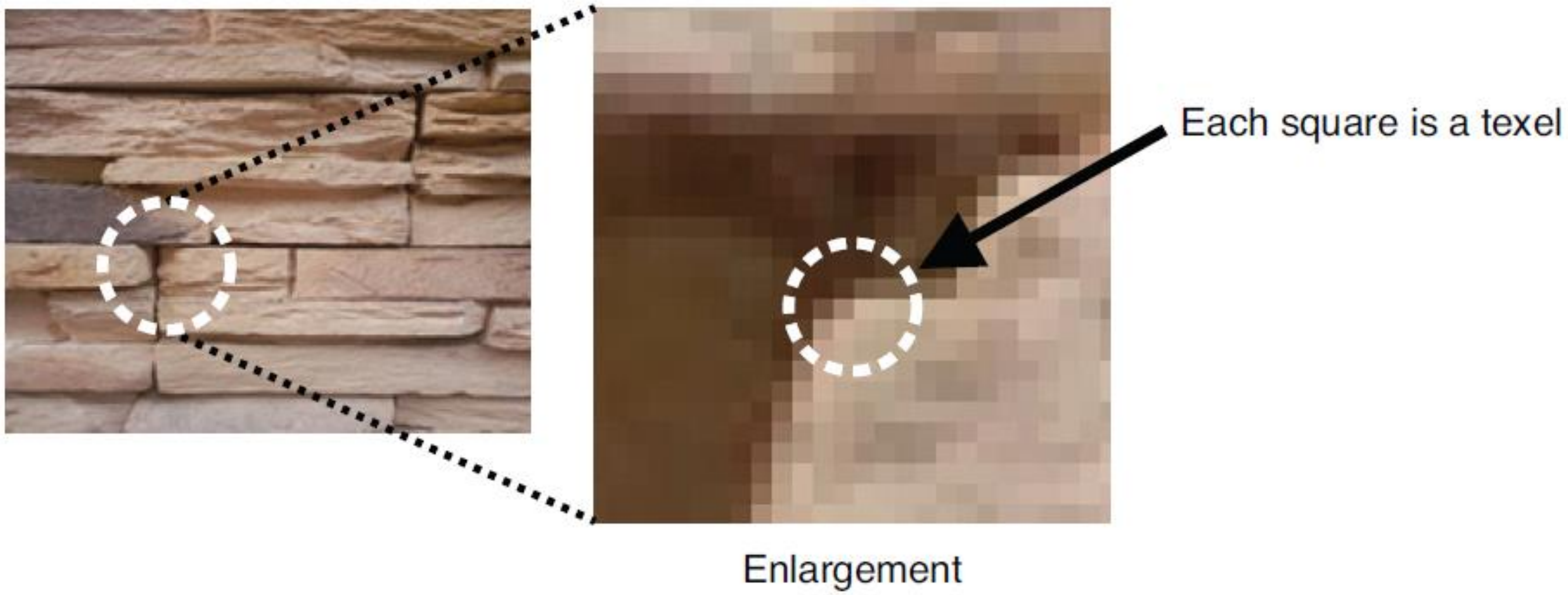


Figure 5.18 Texels

2D Texture Mapping

- However, at this point, we prefer to think of this array as a continuous rectangular two-dimensional texture pattern *T(s, t)*. The independent variables *s* and *t* are known as **texture coordinates**. With no loss of generality, we can scale our texture coordinates to vary over the interval [0.0, 1.0].

2D Texture Mapping

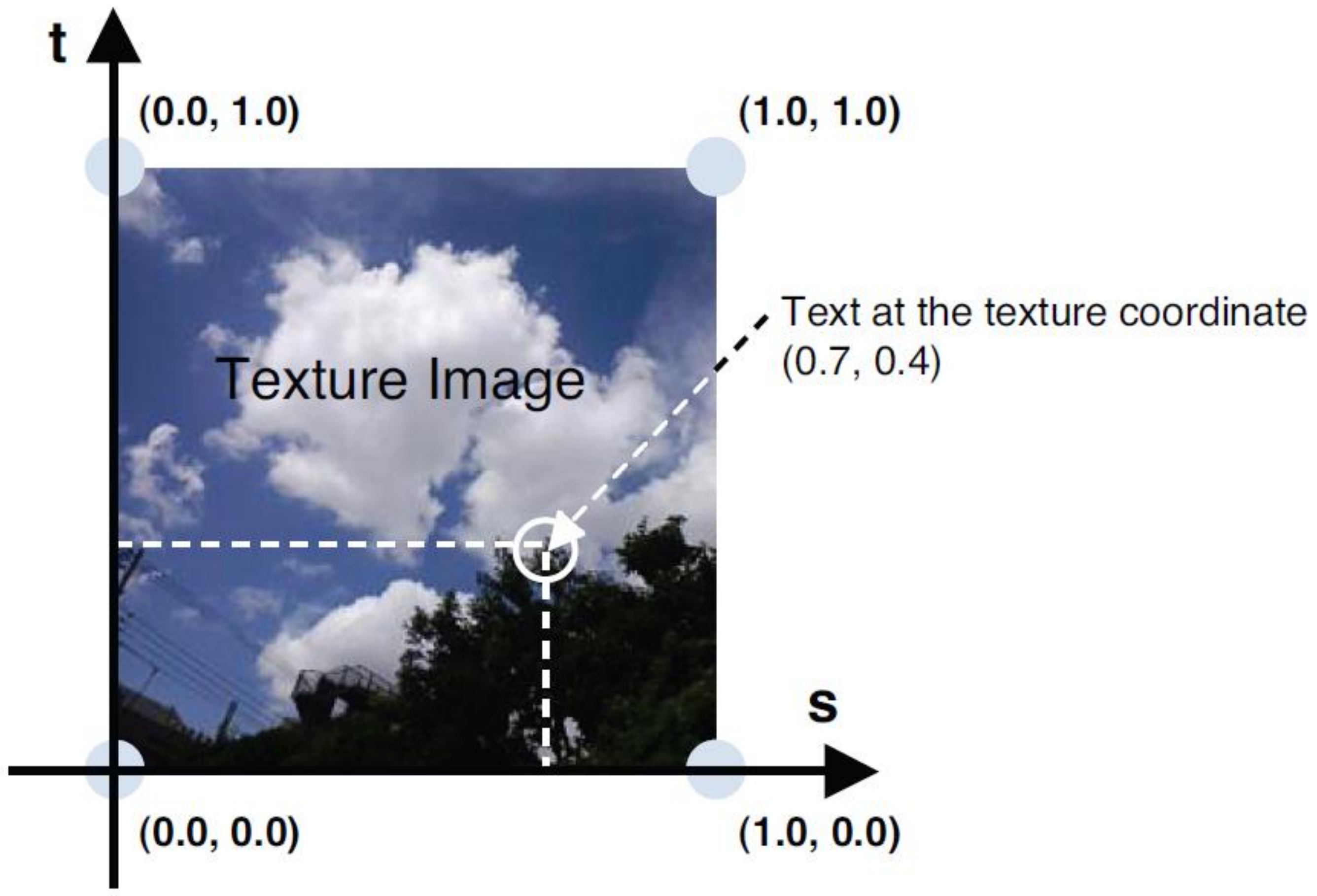


Figure 5.20 WebGL's Texture coordinate system

2D Texture Mapping

- A **texture map** associates a texel with each point on a geometric object that is itself mapped to screen coordinates for display. If the object is represented in homogeneous or (*x*, *y*, *z*, *w*) coordinates, then there are functions such that

$$\begin{aligned} x &= x(s,t) \\ y &= y(s,t) \\ z &= z(s,t) \\ w &= w(s,t). \end{aligned}$$

2D Texture Mapping

- By defining the correspondence between the texture coordinates and the vertex coordinates of the geometric shape, you can specify how the texture image will be pasted.

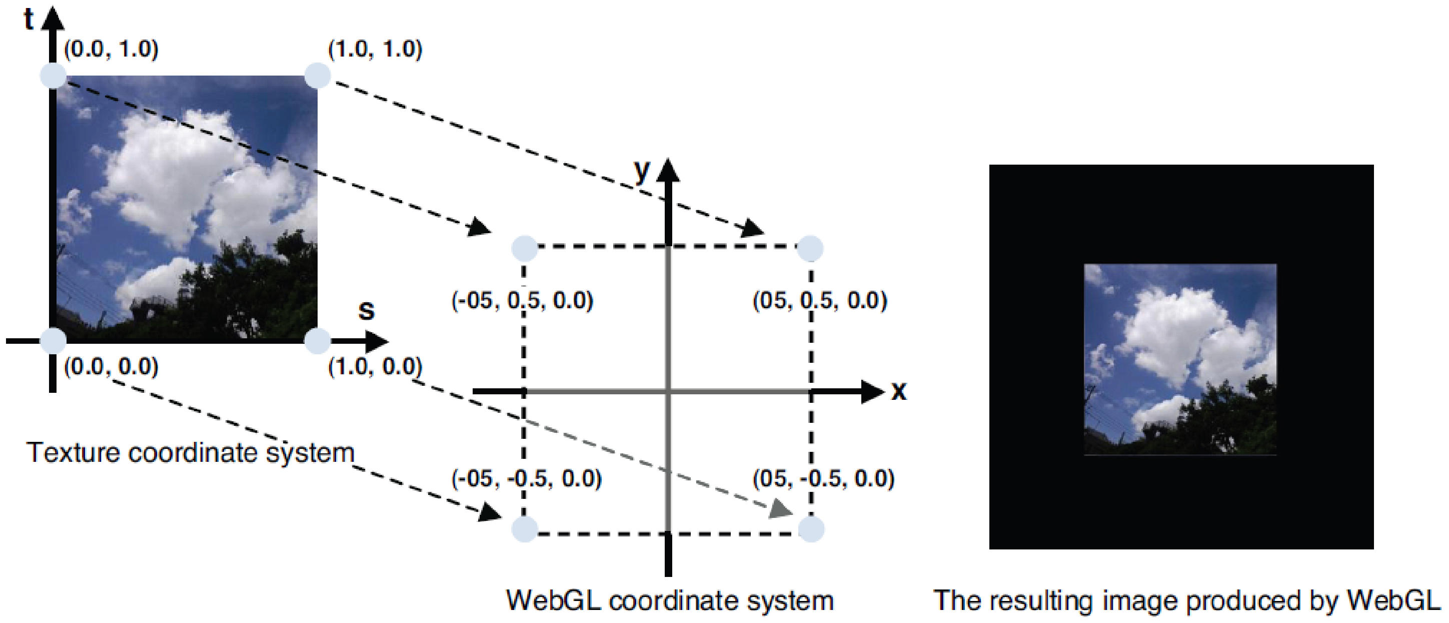


Figure 5.21 Texture coordinates and mapping them to vertices

2D Texture Mapping

- One of the difficulties we must confront is that although these functions exist conceptually, finding them may not be possible in practice. In addition, we are worried about the inverse problem: given a point (*x*, *y*, *z*) or (*x*, *y*, *z*, *w*) on an object, how do we find the corresponding texture coordinates, or equivalently, how do we find the "inverse" functions

$$\begin{aligned} s &= s(x, y, z, w) \\ t &= t(x, y, z, w) \end{aligned}$$

to use to find the texel *T(s*, *t*)?

2D Texture Mapping

- If we define the geometric object using parametric (*u*, *v*) surfaces, as we did for the sphere, there is an additional mapping function that gives object coordinate values (*x*, *y*, *z*) or (*x*, *y*, *z*, *w*) in terms of *u* and *v*. We also need the mapping from parametric coordinates (*u*, *v*) to texture coordinates and sometimes the inverse mapping from texture coordinates to parametric coordinates.

2D Texture Mapping

- We also have to consider the projection process that takes us from object coordinates to screen coordinates, going through eye coordinates, clip coordinates, and window coordinates along the way. We can abstract this process through a function that takes a texture coordinate pair (*s*, *t*) and tells us where in the color buffer the corresponding value of *T(s*, *t*) will make its contribution to the final image. Thus, there is a mapping of the form

$$\begin{aligned} x_s &= x_s(s,t) \\ y_s &= y_s(s,t) \end{aligned}$$

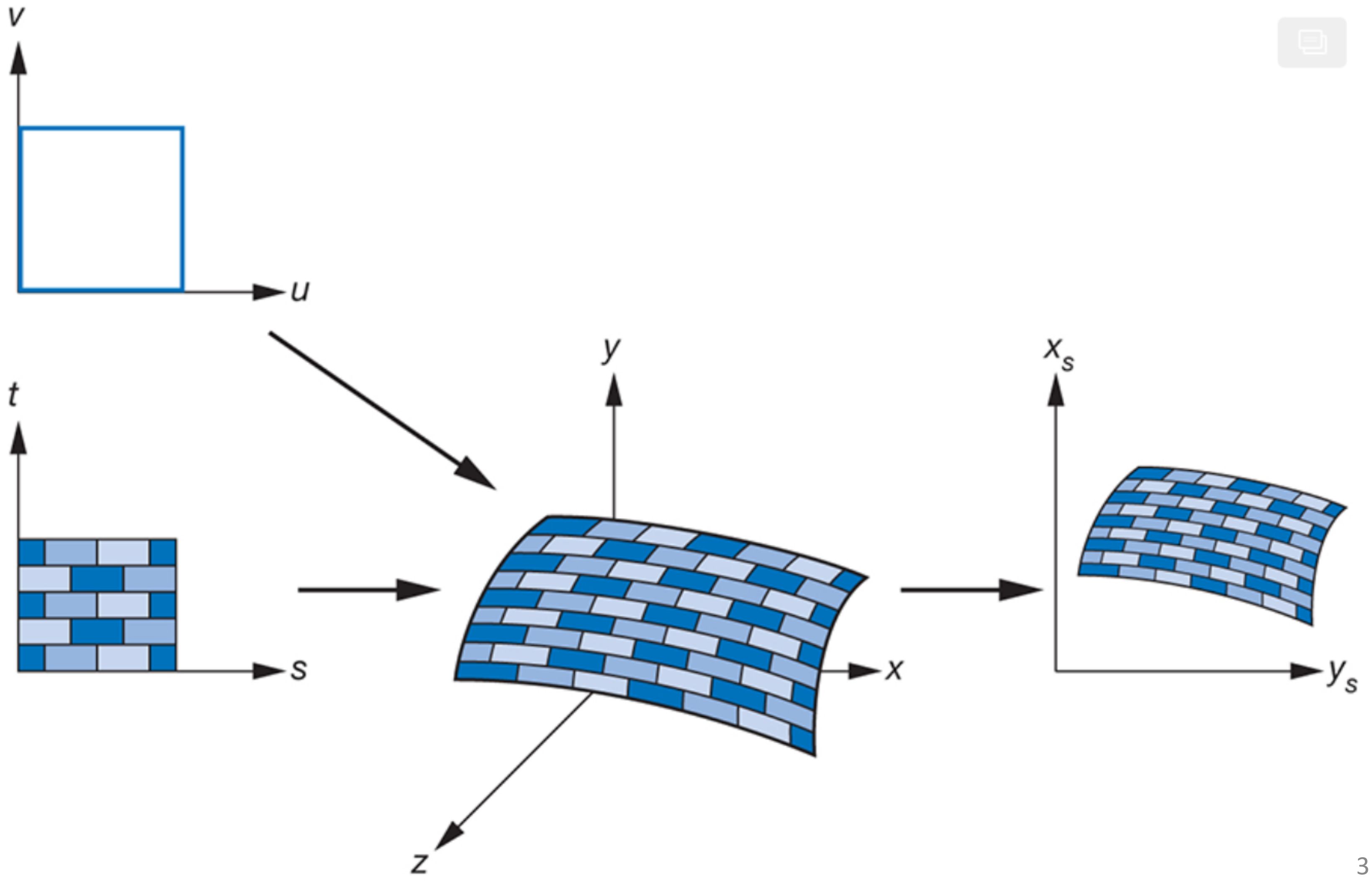
into coordinates, where (x_s, y_s) is a location in the color buffer.

2D Texture Mapping

- Depending on the algorithm and the rendering architecture, we might also want the function that takes us from a pixel in the color buffer to the texel that makes a contribution to the color of that pixel.

2D Texture Mapping

Figure 7.5 Texture maps for a parametric surface.



2D Texture Mapping

- Conceptually, the texture-mapping process is simple.
- A small area of the texture pattern maps to the area of the geometric surface, corresponding to a pixel in the final image.
- This color assignment is carried out as part of the computation of fragment colors.

2D Texture Mapping

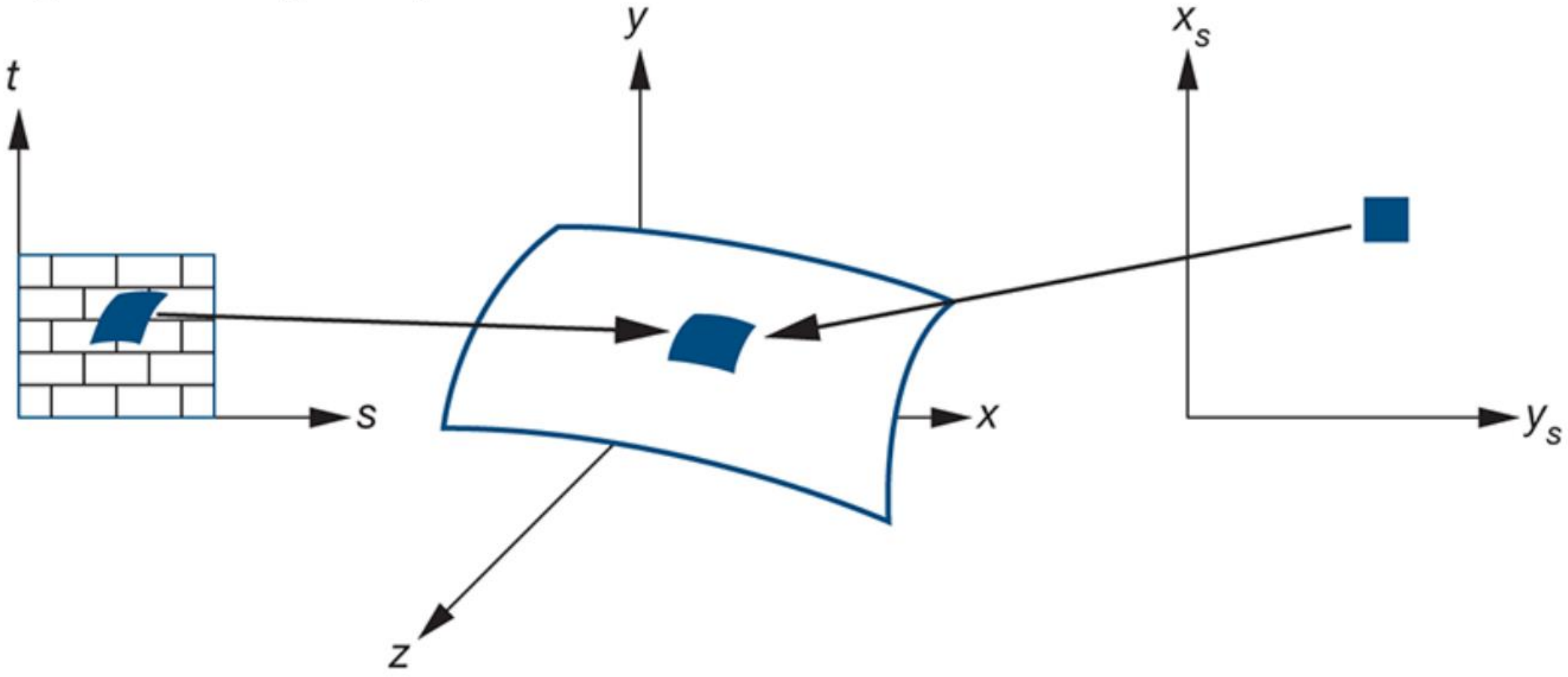
- On closer examination, we face some difficulties.
- We must determine the map from texture coordinates to object coordinates. A two-dimensional texture usually is defined over a rectangular region in texture space. The mapping from this rectangle to an arbitrary region in three-dimensional space may be a complex function or may have undesirable properties. For example, if we wish to map a rectangle to a sphere, we cannot do so without distortion of shapes and distances.

2D Texture Mapping

- We are more interested in the inverse map from screen coordinates to texture coordinates. When we are determining the shade of a pixel, we must determine what point in the texture image to use—a calculation that requires us to go from screen coordinates to texture coordinates.
- Because each pixel corresponds to a small rectangle on the display, we are interested in mapping not points to points, but rather areas to areas. Here is a potential aliasing problem that we must treat carefully if we are to avoid artifacts.

2D Texture Mapping

Figure 7.6 Preimages of a pixel.

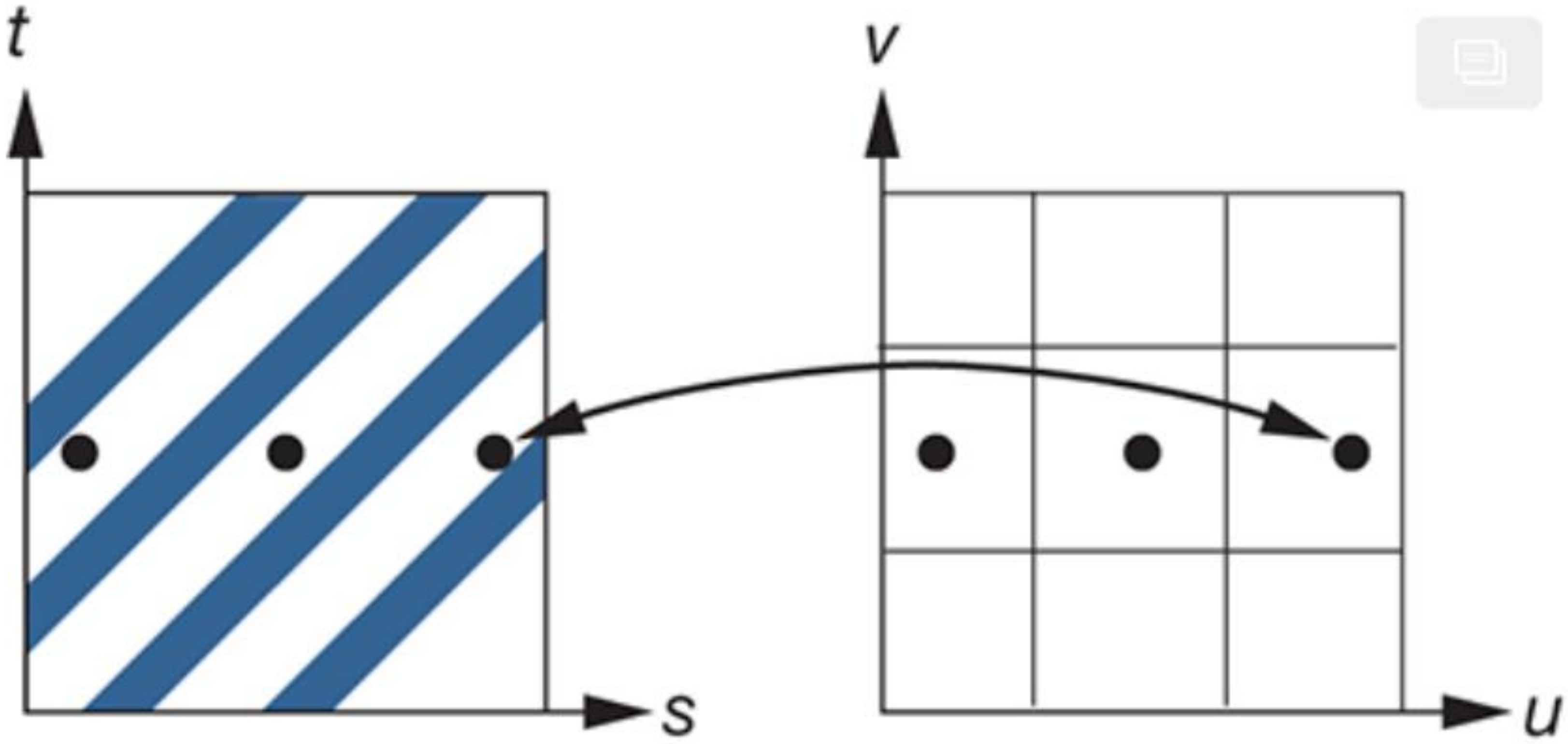


2D Texture Mapping

- Let's put aside for a moment the problem of how we find the inverse map and look at the determination of colors.
- One possibility is to use the location that we get by back projection of the pixel center to find a texture value. Although this technique is simple, it is subject to serious aliasing problems, which are especially visible if the texture is periodic.

2D Texture Mapping

Figure 7.7 Aliasing in texture generation.



2D Texture Mapping

- Now we turn to the mapping problem. In computer graphics, most curved surfaces are represented parametrically. A point **p** on the surface is a function of two parameters *u* and *v*. For each pair of values, we generate the point

$$\mathrm{P}(u,v) = \begin{bmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{bmatrix}.$$

2D Texture Mapping

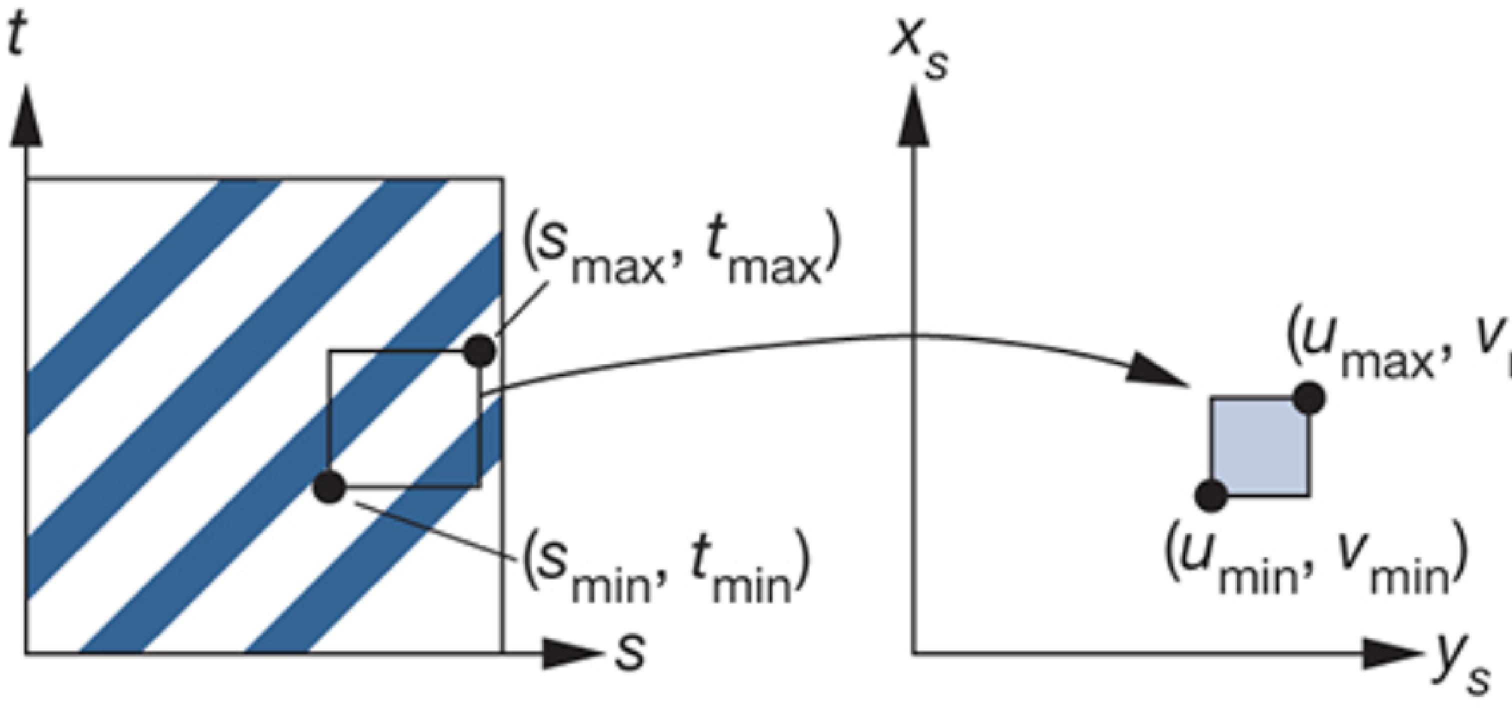
- Given a parametric surface, we can often map a point in the texture map *T(s*, *t*) to a point on the surface **p**(*u*, *v*) by a linear map of the form

$$\begin{aligned} u &= as + bt + c \\ v &= ds + et + f. \end{aligned}$$

As long as *ae≠bd*, this mapping is invertible. Linear mapping makes it easy to map a texture to a group of parametric surface patches.

2D Texture Mapping

Figure 7.8 Linear texture mapping.



$$\begin{aligned} u &= u_{\min} + \frac{s - s_{\min}}{s_{\max} - s_{\min}}(u_{\max} - u_{\min}) \\ v &= v_{\min} + \frac{t - t_{\min}}{t_{\max} - t_{\min}}(v_{\max} - v_{\min}) . \end{aligned}$$

2D Texture Mapping

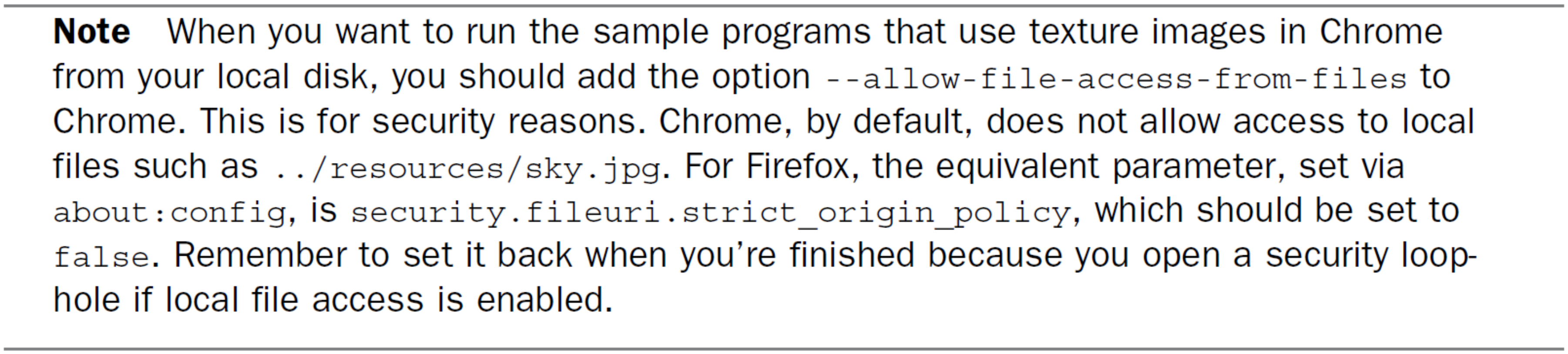
- This mapping is easy to apply, but it does not take into account the curvature of the surface. Equal-sized texture patches must be stretched to fit over the surface patch.

Pasting Texture Images onto the Geometric Shape

Texture mapping involves the following four steps in WebGL:

1. Prepare the image to be mapped on the geometric shape.
2. Specify the image mapping method for the geometric shape.
3. Load the texture image and configure it for use in WebGL.
4. Extract the texels from the image in the fragment shader, and accordingly set the corresponding fragment.

Pasting Texture Images onto the Geometric Shape



Note When you want to run the sample programs that use texture images in Chrome from your local disk, you should add the option `--allow-file-access-from-files` to Chrome. This is for security reasons. Chrome, by default, does not allow access to local files such as `../resources/sky.jpg`. For Firefox, the equivalent parameter, set via `about:config`, is `security.fileuri.strict_origin_policy`, which should be set to `false`. Remember to set it back when you're finished because you open a security loophole if local file access is enabled.

- Use Python HTTP Server
  - Open Command Prompt
  - Go to the folder, then type: python3 -m http.server or python -m http.server
  - In your browser, type: http://localhost:8000/TexturedQuad.html
  - Close the server by Ctrl + c