Texturing, Sampling and Filtering

Sandro Spina
Computer Graphics and Simulation Group

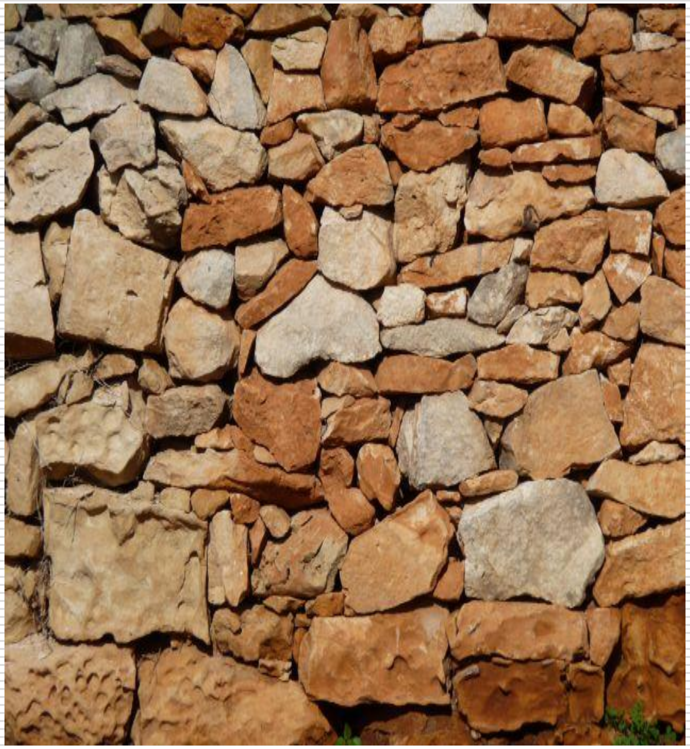
Computer Science Department
University of Malta

Texturing …

- 3D models can be made to look better in a number of ways:
  - Increasing the vertex count
  - Improving the vertex positions
  - Improving the illumination model used for lighting
  - Making use of textures and leveraging visual perception
- 3D models can be made to look better by applying a texture across the polygons composing the model.
- This is the topic of this module. Based mostly on the book Real-Time Rendering (Akenine Mueller)

Texturing … an example

- Suppose we want to model a rubble wall
- We have three options
  - 3D model the different stones then place them individually one on top of the other (very time consuming …)
  - Model the wall surface (this is probably even more complex … because you need to model the spaces between the stones as well)
  - Wrap this image on a 2D surface. (easiest and cheapest)



Generalised Texturing

- Texturing, at its simplest, is a technique for efficiently modelling a surface’s properties.
- One can think of it as wrapping a wallpaper to a surface.
- As seen in previous modules colour is computed by taking into account the lighting and the material together with the viewer’s position.
- *Important:* Texturing works by modifying the values used in the lighting equation.

Change in lighting …

- Recall how the value of a single shaded pixel is calculated from the lighting equation.
- It is computed by taking into account the lighting, the material and the viewer's position.
- Note - one can think of the texture as one of the properties of a material.

- Texturing works by modifying the values used in the shading equation at a particular location according to the corresponding pixel values found in the texture image of the material.

- **Texels:** refers to the pixels in the image texture …

The brick wall example (multiple textures)

- Apart from the lack of geometry details (i.e. attaching the texture to a flat surface) there might be other components which can look unconvincing.

- For Example:
    - The glossiness of the mortar might be different – the bricks should probably be more matte than the mortar.
        - Solution: Apply a secondary texture that changes the glossiness in different parts of the surface

    - When viewed closely bricks appear to be flat !!
        - Solution: Apply bump mapping in order to give the impression that the bricks are not perfectly smooth

Image from pg148 RTR

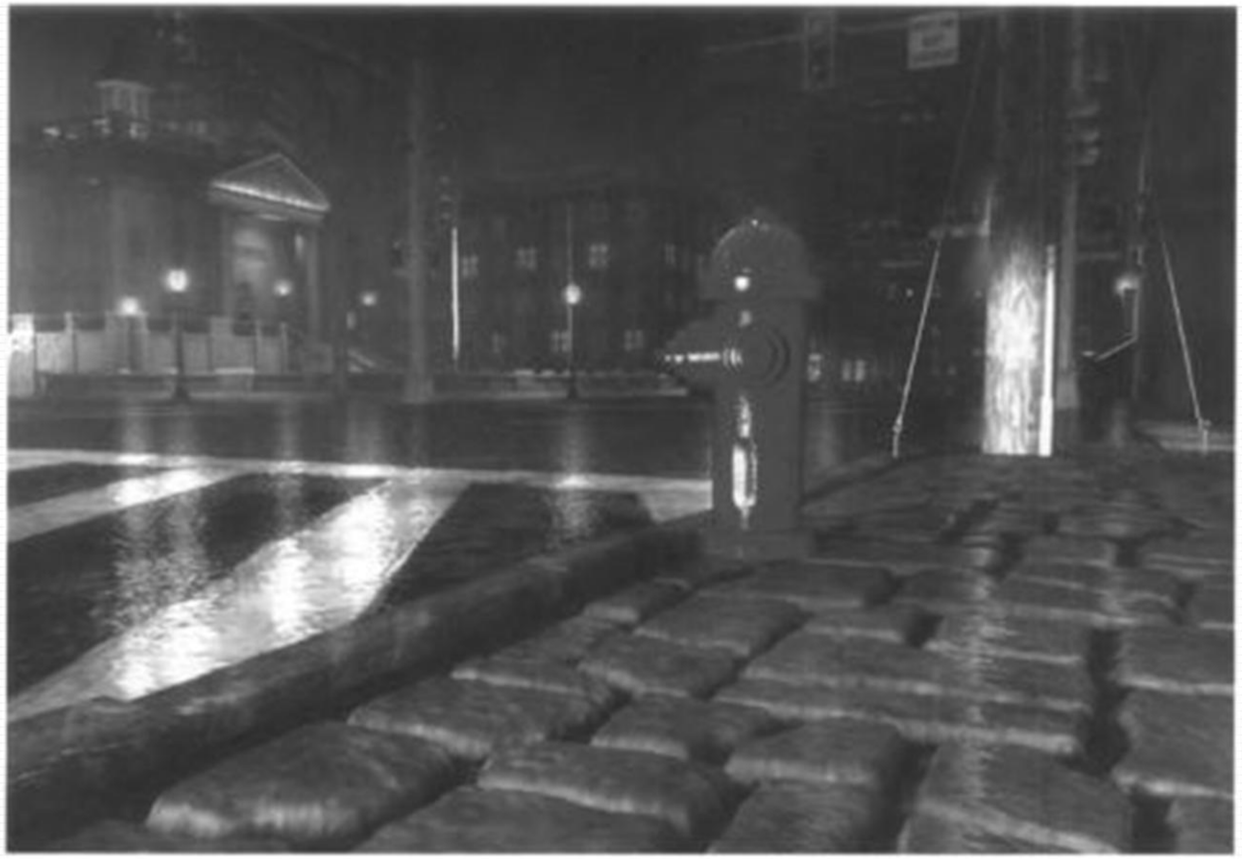
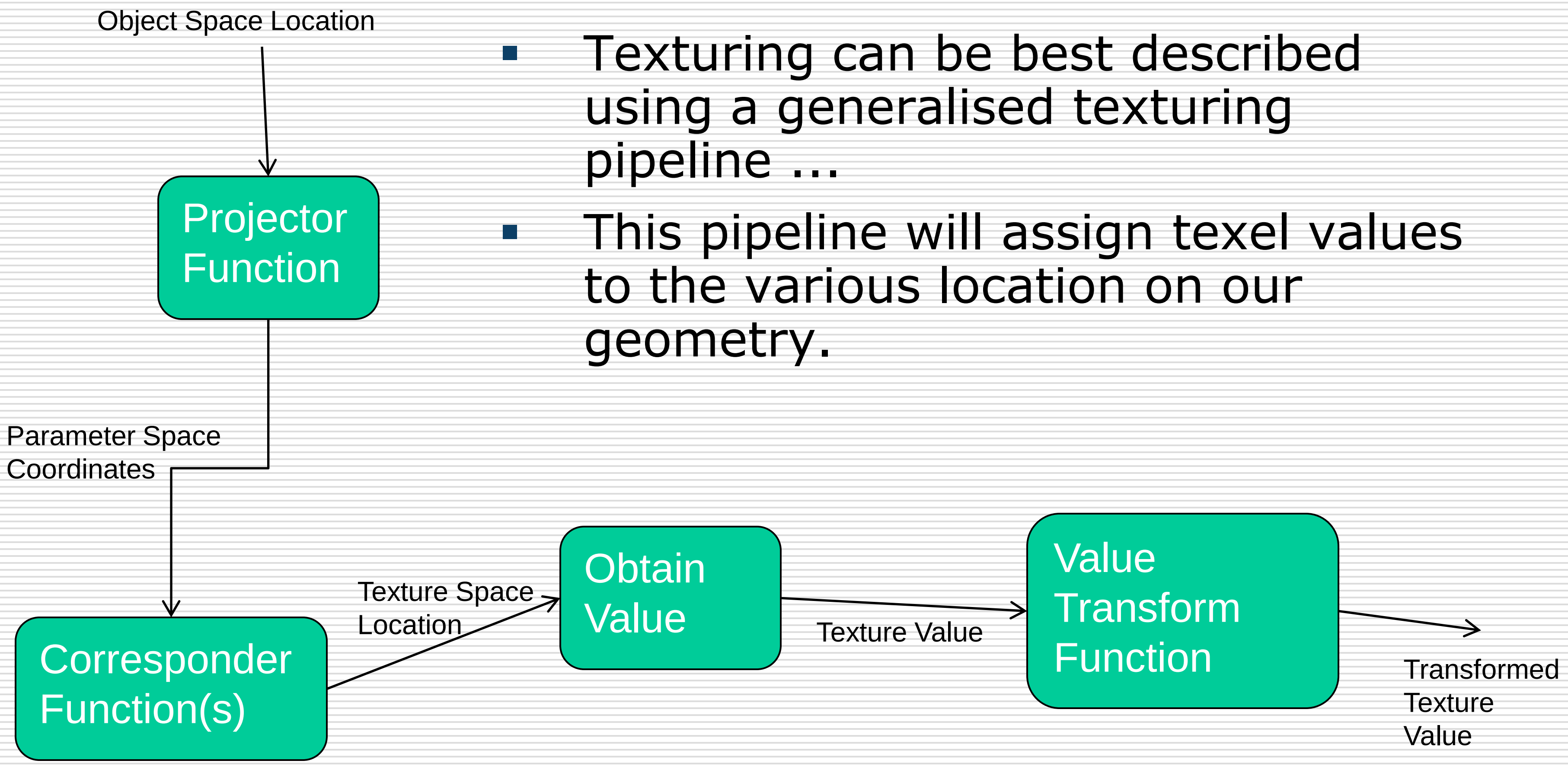


Figure 6.1. Texturing. Color, bump, and parallax occlusion texture mapping methods are used to add complexity and realism to a scene. *(Image from "Toyshop" demo courtesy of Natalya Tatarchuk, ATI Research, Inc.)*

The Texturing Pipeline (4 phases)



- Texturing can be best described using a generalised texturing pipeline …
- This pipeline will assign texel values to the various location on our geometry.

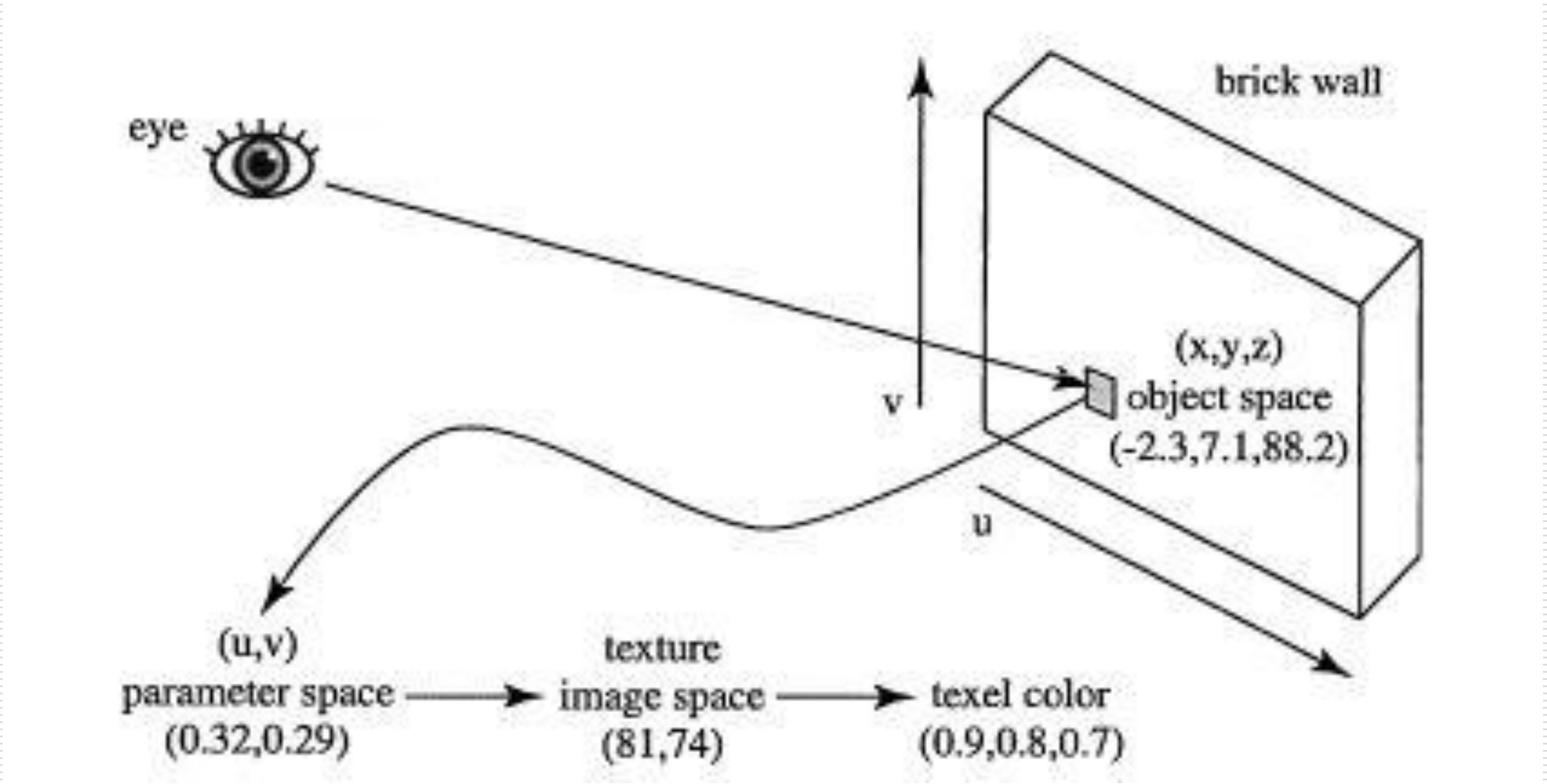
The Texturing Pipeline (4 phases) [i]

- The **initial input** to the pipeline is a location in space (x,y,z)
- This space can be either world space or object space
- Using the model's frame of reference is better (and easier to work with) ... You can think of the texture as moving along with the model.
- Textures are 2D hence this input (point in space) needs to be projected onto it. This is done by using a *projector* function applied to it to obtain a set of values.
- These are called the **parameter-space values**, that will be used for accessing the texture.

The Texturing Pipeline (4 phases) [ii]

- The process of obtaining these parameter-space values is called mapping (the term texture mapping comes from here)
- Before these parameter-space values are used to access the texture, one or more **corresponder functions** can be applied to transform these values into texture space.
- These texture space locations are then used to access the pixel value required.
- Finally, the retrieved values (from the texture) are then potentially transformed again by a **value transform** function.
- These values are then used to modify (material, normal, etc) the surface being rendered.

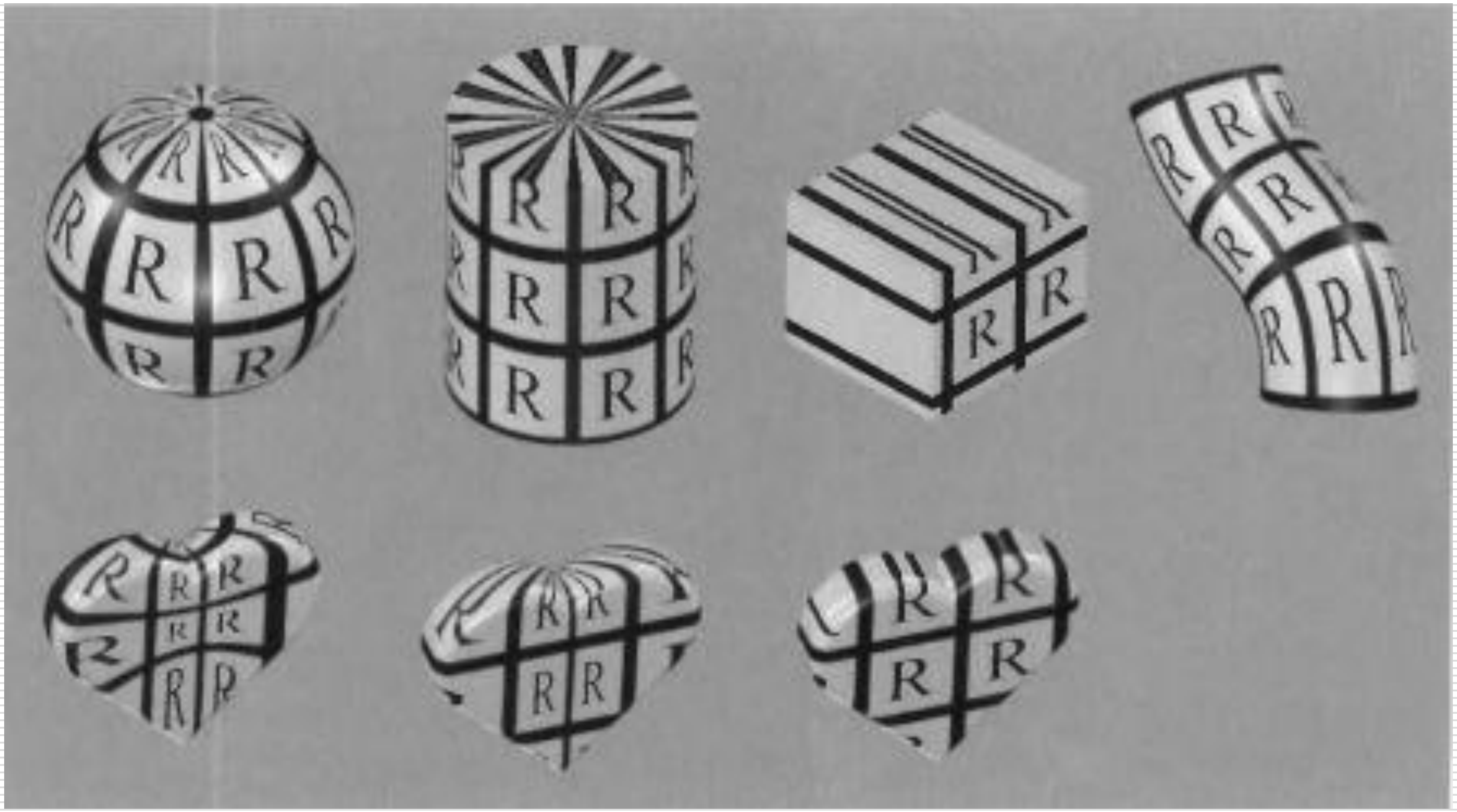
Pipeline for brick wall (Image from RTR)



The Projector Function

- Given a surface (of an object) location, we project this point into parameter space.
- This space is usually (during this course always) a two-dimensional (u,v) space.
- This mapping can either be done
  - Manually by modelling artists using some modelling software. In this case each vertex is assigned a (u,v) value. Note that one can always use a projector function to initialise these values. These can be changed later.
  - Automatically through a projector function
    - Spherical
    - Cylindrical
    - Planar
- Note however that a spherical projection does not need to always be carried out on a sphere. (check next slide)

Different Projections (Image from RTR)

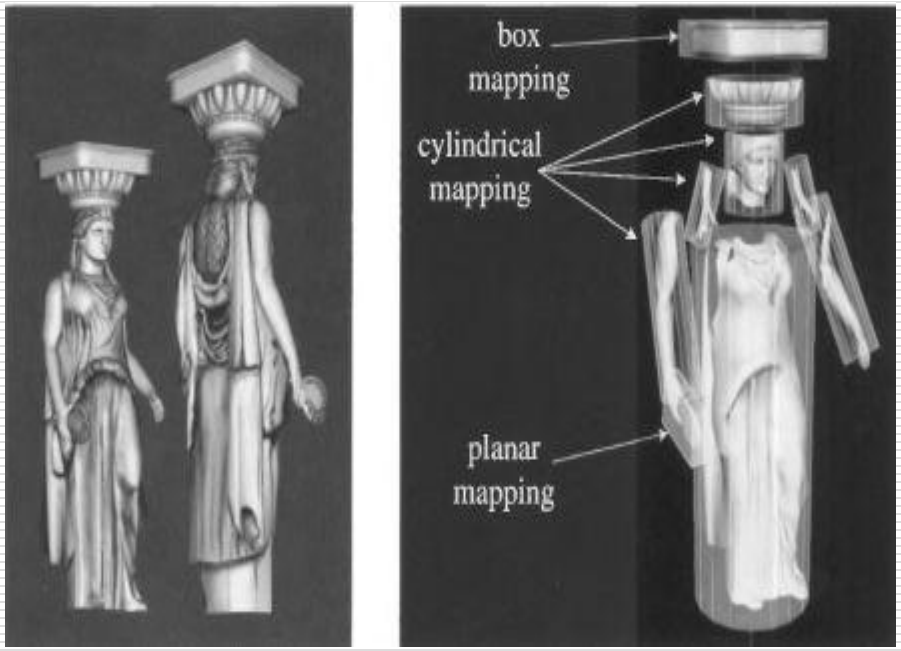


Natural Projections …. And others

- Some projector functions are not projections, but are an implicit part of a surface description.
- Hence if you are rendering a parametric curved surface, then the parameters themselves give you the (u,v) parameter space coordinates.
- The location in space for the object depends on (u,v) hence the texture value is easily found by simply using the same (u,v) parameters.
- Check example in previous slide.
- Texture coordinates can also be generated from other different parameters such as view direction, temperature of surface, etc. As far as you are concerned the projector function will give you texture coordinates.

In most real-time work … (in games for example)

- Projector functions are pre-compiled and stored per vertex at the modelling stage.
- This is especially true for complex geometry
- However there might be situations where the projector function has to be applied in real-time. With environment mapping for example.



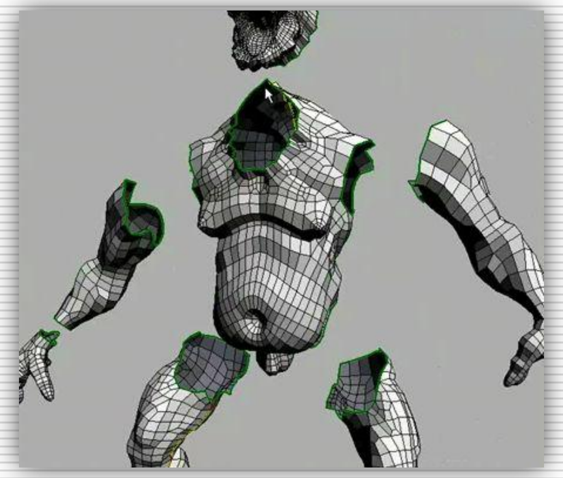
Texture Flattening .... (using UVLayout)

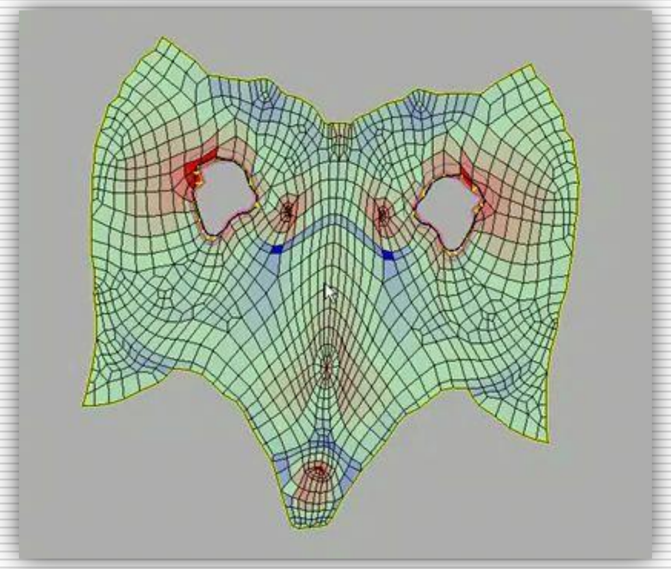
- Once you've got your geometry done we usually use a UV layout package in order to texture our geometries.

- This is especially true for complex geometry

- On your right is an example from the software UVLayout ... The texture map corresponds to the torso of the ogre ... Flattened.

- The holes represent the edges which connect to the arms





The Corresponder Function – Use

- The projector function is applied on the different vertices defining the 3D object.
- In between vertices textures still need to be applied i.e. The parameter values need to be interpolated across the surface and used to retrieve texture values.
- However before being interpolated these parameter values are **transformed** using a *corresponder* function. (recall the example of the brick wall a couple of slides ago)
- A corresponder function converts parameter space coordinates into texture space coordinates.
- Different corresponder functions provide adequate functionality and flexibility when applying textures to surfaces. We'll check some in the next slide.

The Corresponder Function – Examples

- Various corresponder function have been defined:
  - Texture portion selection – suppose you only want to use part (a sub image) of the available texture.
  - Transformation matrix – this corresponder function can be used to perform rotations, scaling, etc on the texture.
  - Another class of corresponder functions controls the way an image is applied to a surface when the (u,v) parameter space coordinates are beyond [0,1].
  - NOTE IMPORTANT: that the reason why (u,v) coordinates are specified between [0,1] is mainly related to the fact that we can use different resolution images for the textures. At the vertices we need only store the same value irrespective of resolution of the image.

The Corresponder Function – RTR Image

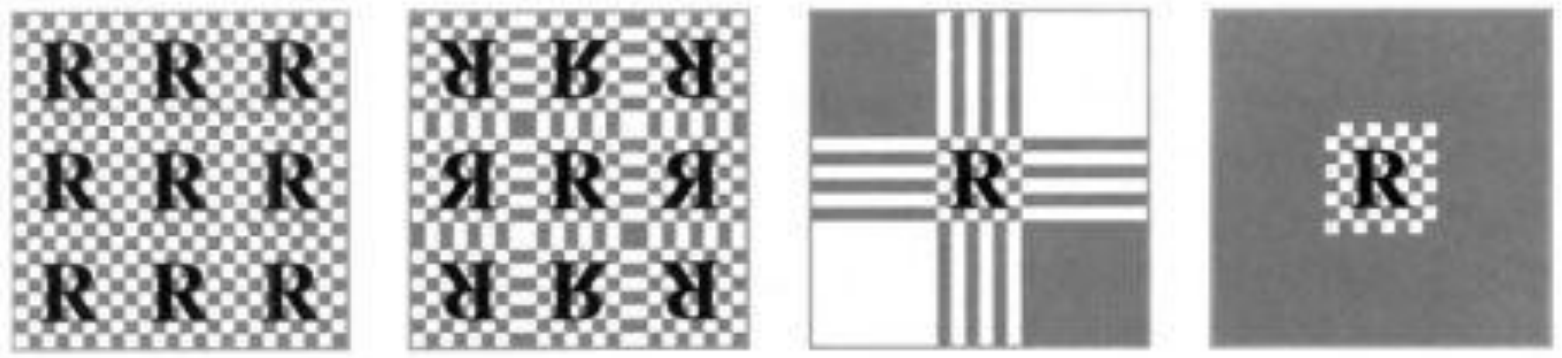


Figure 6.7. Image texture repeat, mirror, clamp, and border functions in action.

The Corresponder Function outside [0,1]

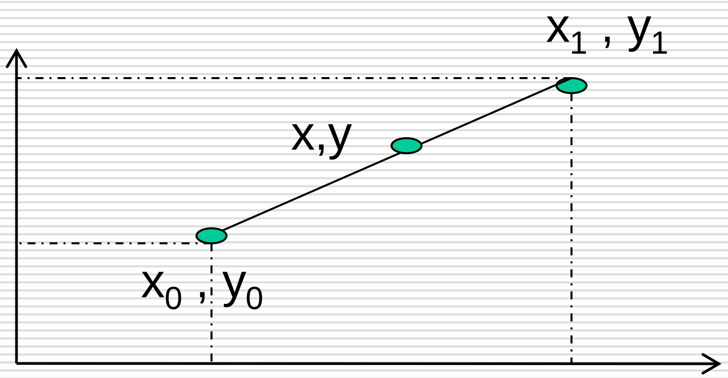
- As seen in the previous slide a number of possibilities are commonly employed to tackle parameterised values outside [0,1]. These are:
  - *Wrap or Repeat*: The image repeats itself across the surface; algorithmically the integer part of the parameter space values are dropped. If you have a tiled surface is will be a good option. Also good for terrains (grass for eg)
  - *Mirror*: The image repeats itself across the surface but is mirrored on every repetition. Good if you want to provide continuity between the edges of the texture on the surface.
  - *Clamp*: Values outside the range [0,1] are clamped to this range ... Essentially extending the edges of the texture on the surface.
  - *Border*: Parameter space values outside the [0,1] range are assigned a predetermined border colour value.
- Note that these functions can be applied differently along both axis ... For e.g. The texture could repeat along the u-axis and get clamped along the v-axis

Texture Values – Output of the pipeline…

- Once we calculate the texture-space coordinates (using the corresponder function of course !!) we can retrieve the texture value.
- We shall only be dealing with image texturing ... In which case the value that we need is a texel.
- Which is usually an RGBa value
  - RGB = Red Green Blue channels
  - A = alpha channel which (normally) describes the opacity of the texel. In practice this value will tell us how much the texel colour will effect the surface pixel.
- Optionally this value can be transformed to produce the transformed texture value.
  - This will depend on what sort of texture mapping one is doing. For eg in shadow mapping some transformations need to take place here.

What is Interpolation?

- Linear first !! ...
- Lerp quasi acronym for LI
- Is usually a function within math APIs

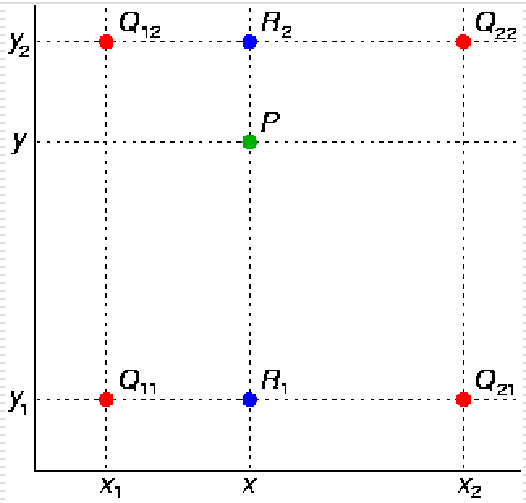


- Given two known points (x0, y0) and (x1, y1) we can interpolate the values of y through x = x0 to x1

$$\frac{y - y_0}{x - x_0} = \frac{y_1 - y_0}{x_1 - x_0}$$

What is Interpolation?

- Bilinear ...
- Linear interpolation in two directions.
- The first LI (horizontal axis) will give us two points R1 and R2.

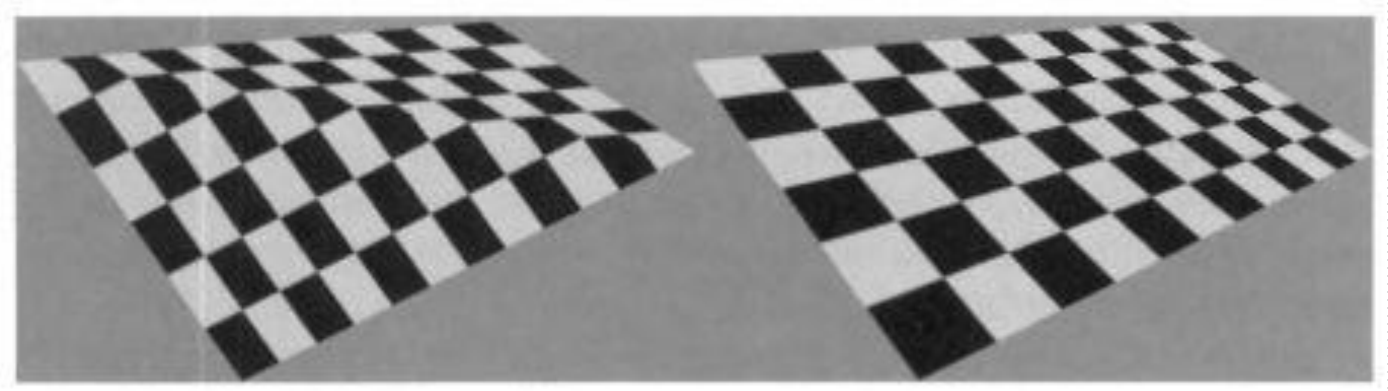


- We then Linearly Interpolate (LI) between R1 and R2 in order to get the required value P.
- You can check the equation on many textbooks and on some sites on the internet.

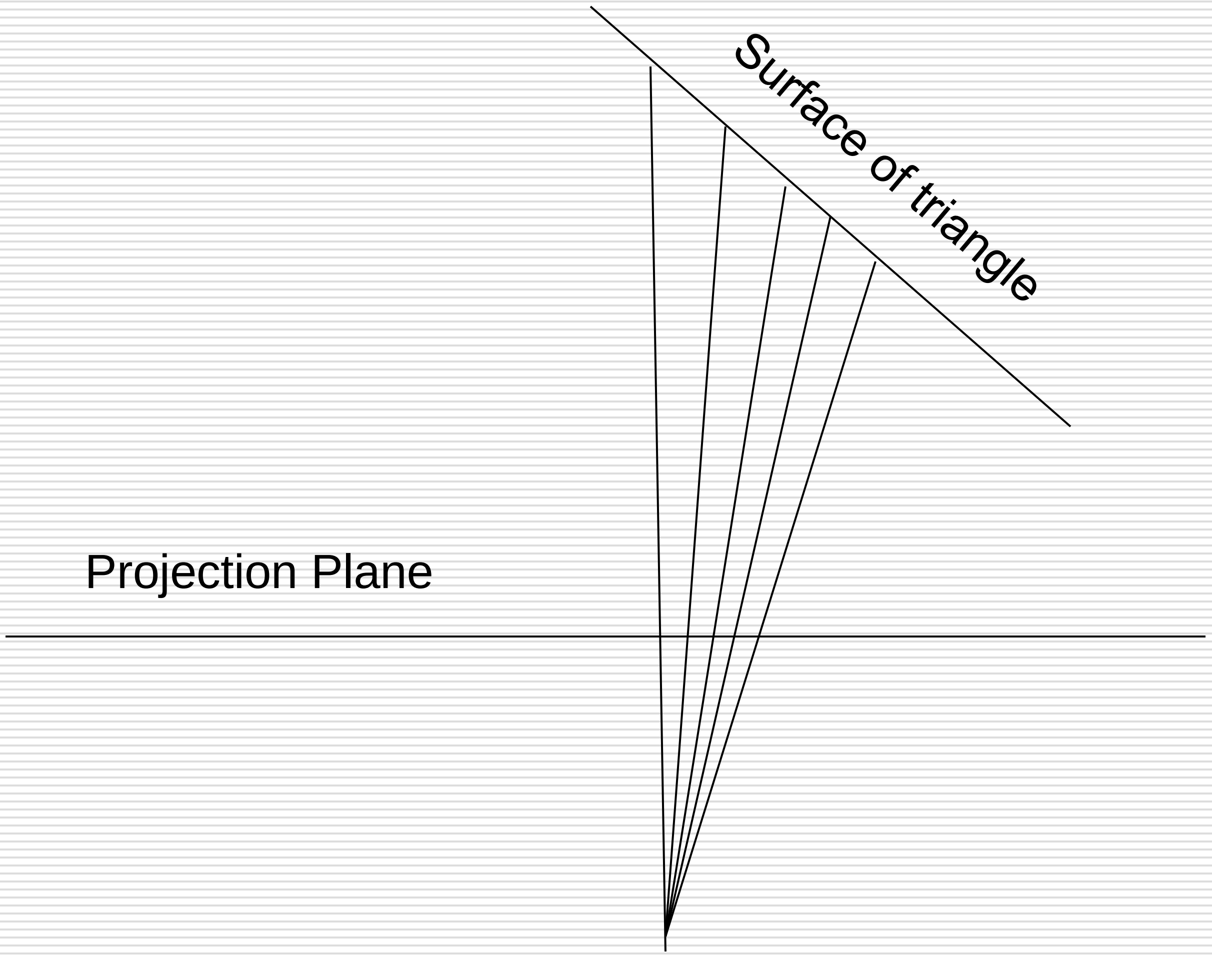
Perspective Correct Interpolation for Texture Mapping on Primitives

- As you know by now rasterisation is the process of converting primitives (traingles, lines, etc) onto the screen, i.e. determining which pixels make up the primitive.

- When mapping textures to these surfaces problems arise on oblique surface if linear interpolation of the textures is carried out as shown in the image below.



Perspective Correct Interpolation (Visual)



Perspective Correct Interpolation (i)

- Suppose we have a vertex, **v**, that is perspectively projected (check viewer optics slides), then divided by the *w* component to obtain the NDC **p** = $(p_x, p_y, p_z, 1)$

- Recall that p_z is now stored in the z-buffer

- Each vertex will have (u,v) coordinates stored as well

- The screen positions can be linearly interpolated … To get the intermediate p values.

- Problem is that if texture coordinates are linearly interpolated, improper foreshortening due to the perspective effect will be achieved (as seen in prev image)

Perspective Correct Interpolation (ii)

- A simple solution which uses the w (depth value) is used as proposed by Heckbert, Moreton and Blinn.
- Instead of linearly interpolated u and v, they demonstrate that we can linearly interpolate 1/w and (u/w, v/w) instead.
- The required (u,v) coordinates are then extracted from the values.
- Basically we have (u,v) = (u/w, v/w) / (1/w)
- Perspective correct interpolation is automatically carried out in hardware during rasterisation.

Image Texturing … Some considerations

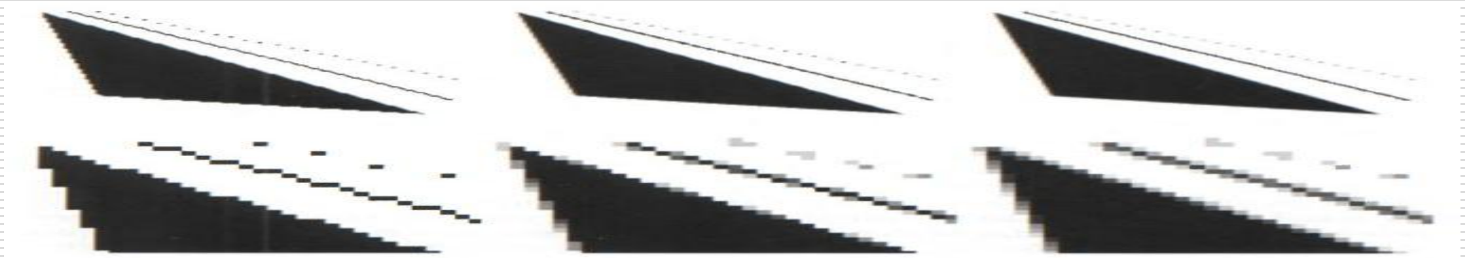
- So far we've seen how to calculate the texture coordinates necessary to retrieve the texel value.
- However things are (as usual) not this simple in practice.
- Assume we have a 256x256 image which we are using as a texture for the side of a box.
- If on screen (when rendered) the box is close to 256x256 pixels then everything will look perfect … You are practically looking at the original image. BUT …
- What happens when the projected square covers much more than 256x256? ***Magnification*** problem.
- What happens when the projected square covers only a few pixels? ***Minification*** problem.

Image Texturing … Magnification and Minification

- At a finer level …
  - Magnification – a single pixel on the screen can correspond to a tiny portion of the texel.
  - Minification – a single pixel no the screen can correspond to a number of texels
- In both cases we need to sort out which texels are going to be used from the texture and how they should be averaged (minification) or interpolated (magnification)
- Aliasing problems can be generated at his stage
- A number of filters can be used (implemented on the GPU) to determine which texel value to use.

Aliasing … and antialiasing

- Aliasing is a visual effect caused whenever the output device does not have enough resolution to output a smooth line/edge.



- The problem here is that the pixel color goes from black to white instantly … there is not grey in between.
- Edges with this stair like feature are referred to as ‘jaggies’
- Antialiasing techniques try to address this problem through the use of sampling and filtering.
- In texture mapping aliasing will occur with minification.

Magnification (example from RTR)

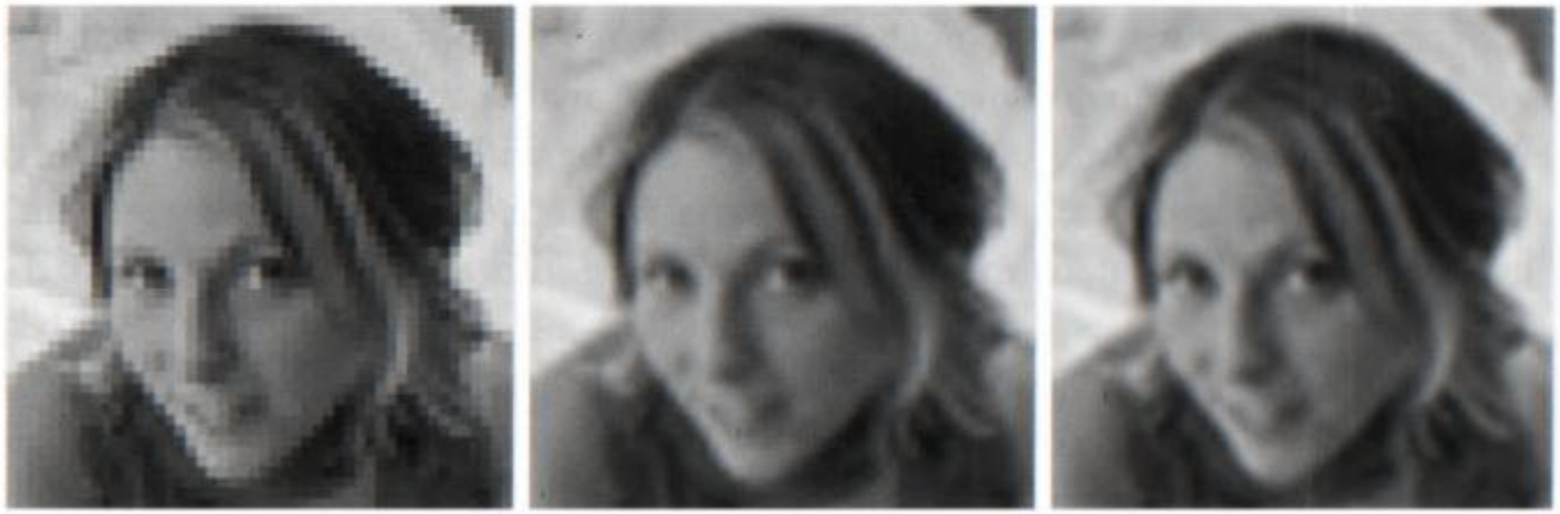


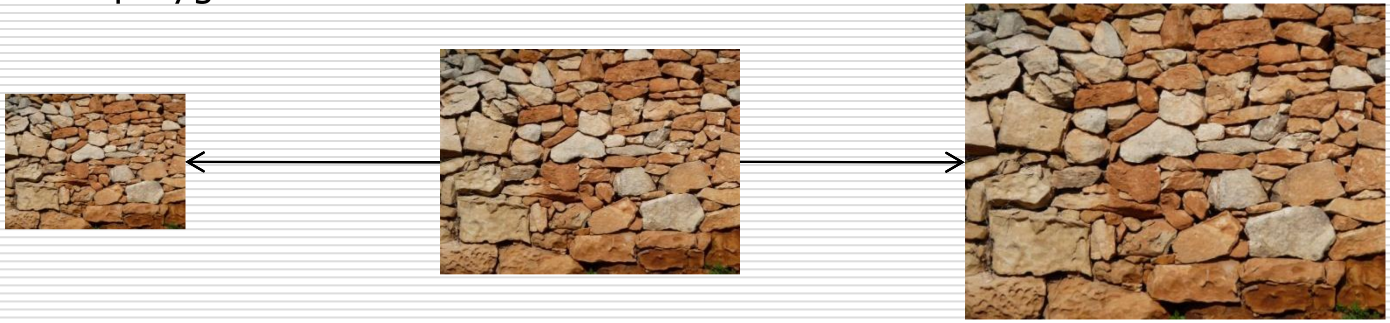
Figure 6.8. Texture magnification of a 48 × 48 image onto 320 × 320 pixels. Left: nearest neighbor filtering, where the nearest texel is chosen per pixel. Middle: bilinear filtering using a weighted average of the four nearest texels. Right: cubic filtering using a weighted average of the 5 × 5 nearest texels.

Before we discuss Magnification and Minification we'll cover the basics of both sampling and filtering.

Sampling …. And Filtering (i)

- The process of rendering an image is inherently a sampling task … given that the generation of a 2D image is the process of sampling a 3D scene in order to obtain colour values for each pixel (This is particulary evident in ray tracing approaches)

- In texture mapping, textures are sampled in order to fill in the space on screen where the polygon stretches.

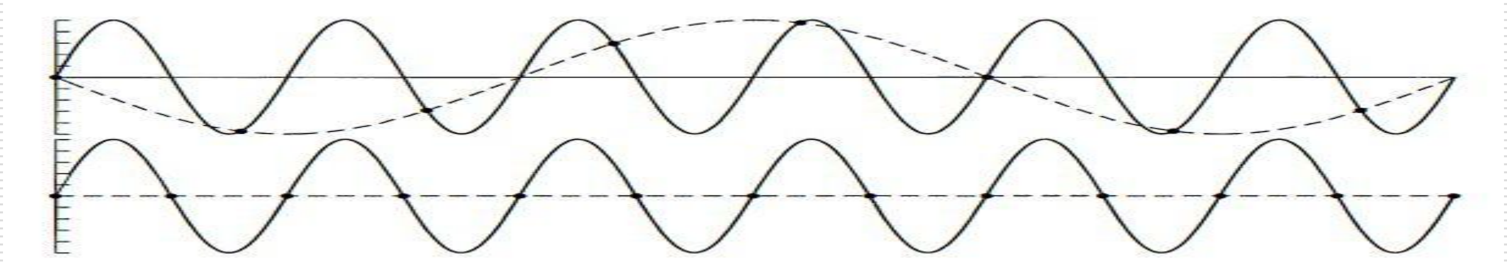


- Let's say the middle image is our texture … to the left and to the right sampling and filtering need to be carried out from the original image (which remember has a fixed number of texels).

- Therefore for texture mapping, texels have to be resampled to get good results under varying conditions.

Sampling …. And Filtering (ii)

- An important concept in sampling theory is the *Nyquist limit*.
- If a signal is undersampled then it's impossible to correctly reconstuct the original signal from the samples …



- i.e. if a texture is undersampled (in minification) then the reconstructed image will not be good enough.
- Aliasing (jaggies) is a common problem resulting from undersampling.

Sampling …. And Filtering (iii)

- In order to alter the size (magnification or minification) of a texture filtering needs to take place.
- The underlying graphics hardware will carry out magnification (or min) using a *filtering* technique.
- The most common filtering techniques are:
  - Box Filter (Nearest Neighbour)
  - Bilinear Interpolation
  - Cubic Convolution
- Convolution is a technique used in order to apply filters and makes use of kernels.
- Filters can be performed in a shader program.

Magnification ...

- In the previous example ... The underlying graphics system needs to enlarge (magnify) the texture.
- This mapping from 48x48 to 320x320 means that some form of filtering needs to occur.
- Box Filter: this filter basically increases the size of each pixel in the original texture. Obviously the individual textures become more apparent, causing an effect referred to as *pixelation.*
- The quality of the filtering method is poor however it's the fastest given that you only need one texel to be fetched per pixel.

Magnification … Bilinear interpolation (i)

- A more commonly used filtering method makes use of bilinear interpolation (across texels)

- For each pixel, four texels (neighbouring ones) are fetched from the texture. These are then interpolated in two dimensions to find a blended value for the new pixel.

- The effect is to blur the image … some sharpness is lost, but also the jaggedness of the box filter has now largely decreased.

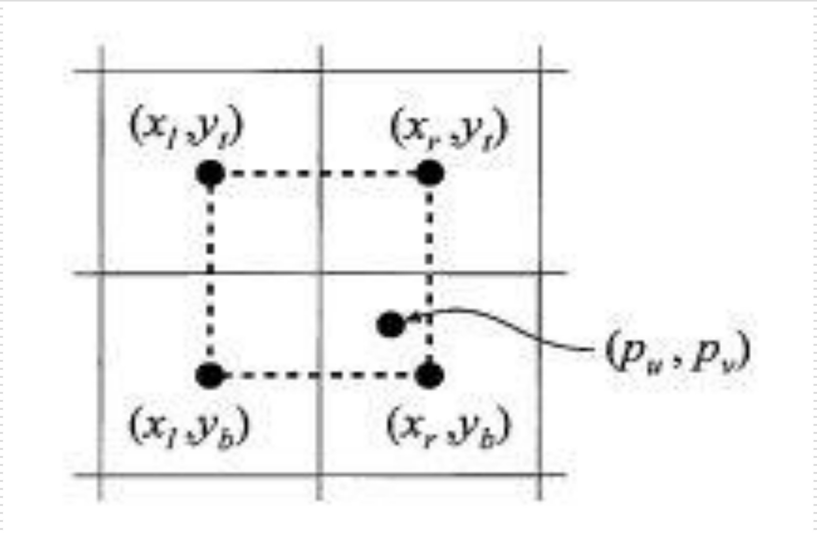
- Let's say $(p_u, p_v) = (81.92, 74.24)$. The fractions are used in computing the bilinear combination of the four closest texels.

- The next slide shows the image (from RTR) describing the notation of bilinear interpolation, together with the formula for $\mathbf{b}(p_u,p_v)$ which gives the bilinearly interpolated value.

Magnification … Bilinear interpolation (ii)

- The interpolation formula becomes simpler if the 4 known points lie on (0,0) (0,1) (1,0) and (1,1).
- $b(p_u, p_v) =$

$(1 - u')(1 - v')\,\mathbf{t}(x_l, y_b) + u'(1 - v')\,\mathbf{t}(x_r, y_b) + (1 - u')v'\,\mathbf{t}(x_l, y_t) + u'v'\,\mathbf{t}(x_r, y_t)$



Magnification … Detail Textures

- Magnification will introduce some form of blurriness in the resulting image …
- One simple technique to alleviate this problem is to use what are referred to as detail textures.
- The techqniue makes use of small (high-resolution) textures such as for example images representing scratches, foliage, or some other surface detail.
- These textures are overlaid repetitively over the original magnified texture as a separate texture, at a different scale.
- This has the effect similar to the use of a single high-resolution texture.

Minification …

- When a texture is minimised, several texels may end up covering the same pixel.
- In order to get a correct colour value one should fetch all the texels which cover the pixel and integrate their effect on the pixel.
- It is effectively impossible to do so in real-time (30/40 times a second) … and most of the times it's a waste of time (suppose there are 1000 texels covering 1 pixel)
- For real-time a number of techniques are used to address this problem.
- In a similar fashion to magnification … a nearest neighbour technique can be used which returns one single texel which is at the centre of the pixel.
- This filter is clearly not very good … it will cause severe aliasing problems as seen in the image on the next slide.

Minification … example

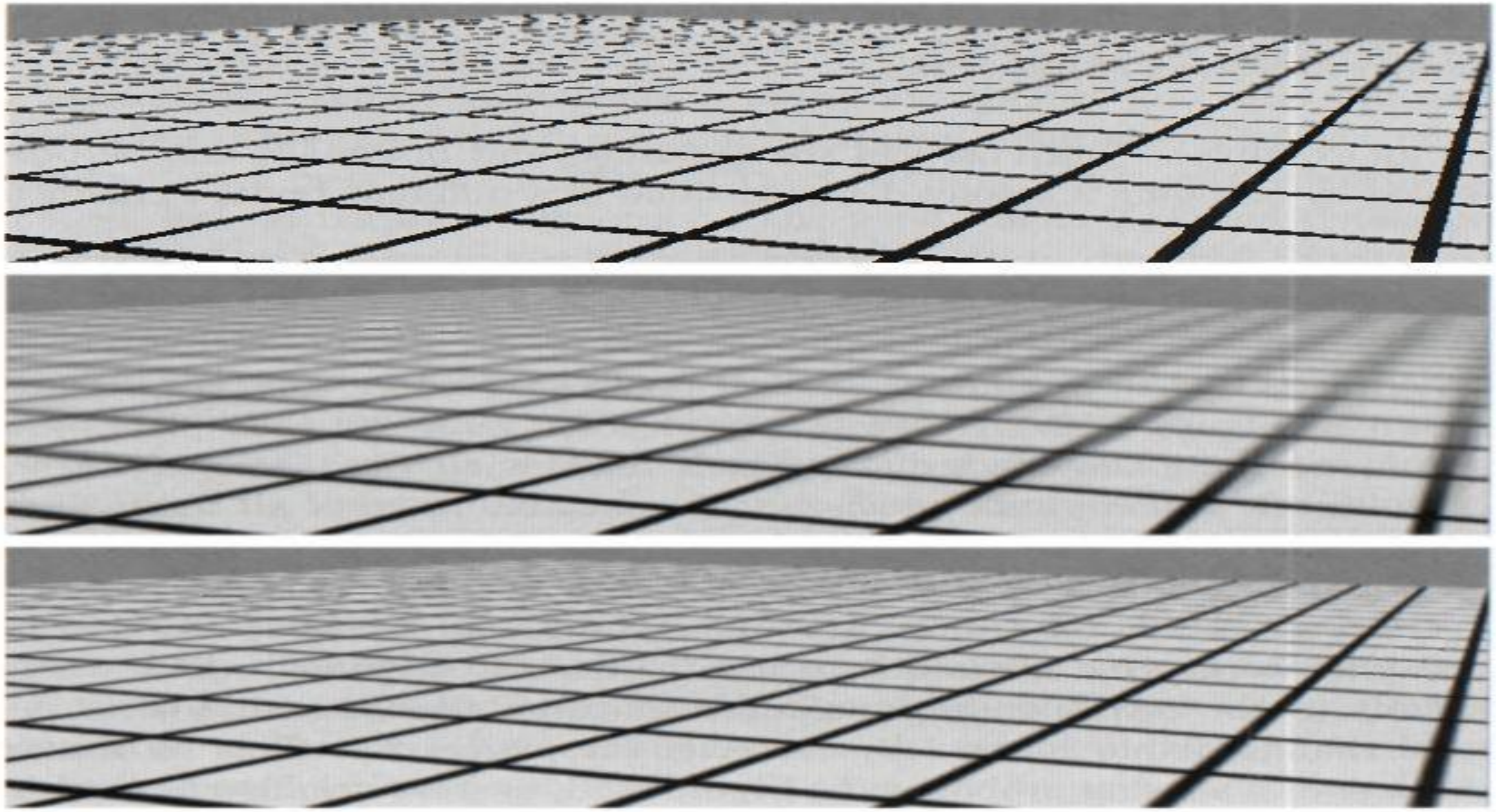


Figure 6.13. The top image was rendered with point sampling (nearest neighbor), the center with mipmapping, and the bottom with summed area tables.

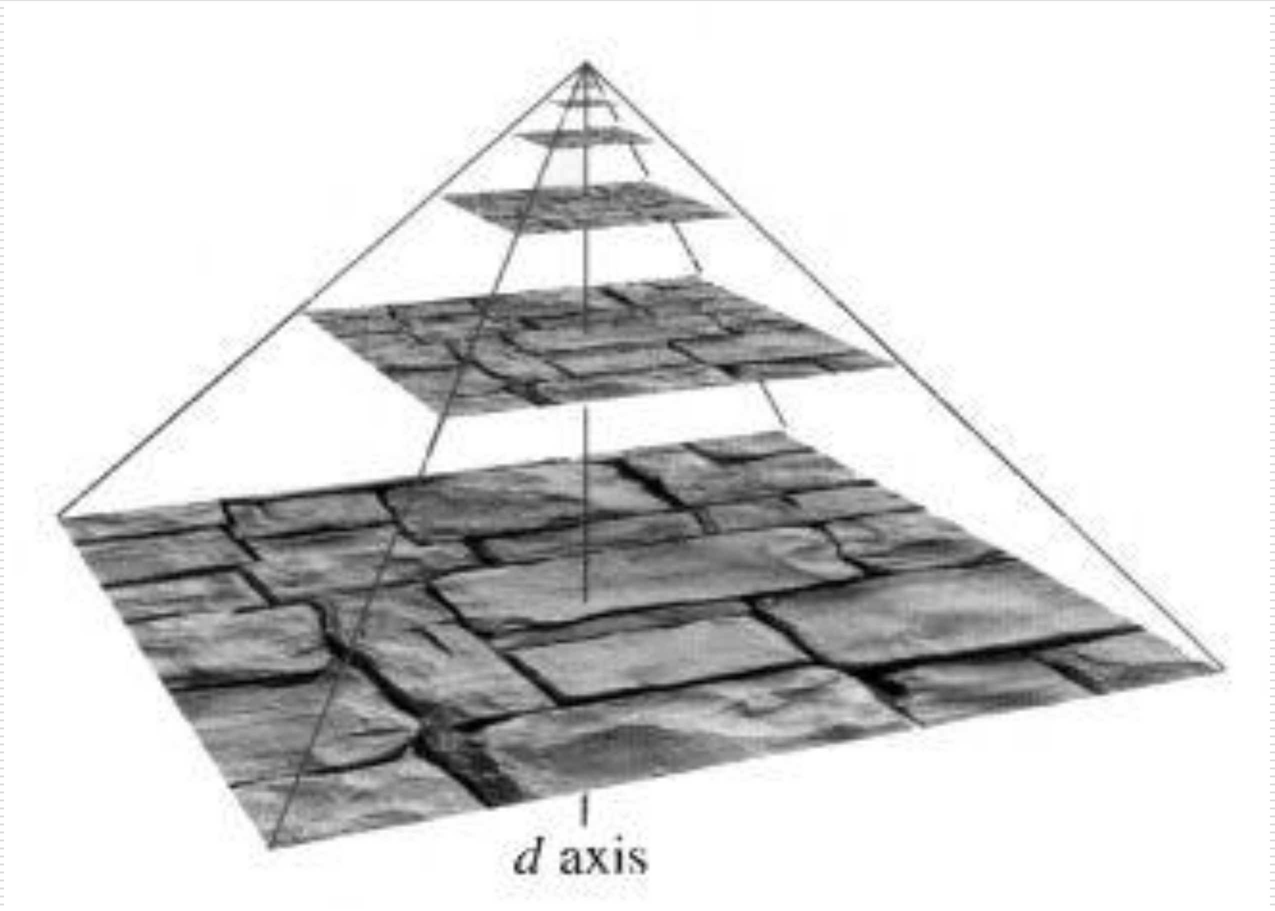
Minification … Bilinear interpolation

- Bilinear interpolation can also be used for minification.
- 4 texels in the original texture are interpolated to produce a value for one pixel.
- Problem here (as opposed to magnification) is that many times a pixel is influenced by more than four texels.
- If this is the case the same problems of a box filter will pop up … i.e. increase in aliasing artefacts.
- Various texture minification algorithms have been developed for real-time work, most of them based on the creation of a data structure which is used to store pre-processed textures purposely built to help compute a quick approximation of the effect of a set of texels on a pixel.

Mipmapping

- A widely used (implemented on all modern GPUs) method of antialiasing for textures (note that antialiasing can be screen based) is called mipmapping.
- MIP stands for *multum in parvo (many things in a small place - minification)*
- Mipmapping takes the original texture and filters it down repeatedly into smaller images – i.e. the original texture is augmented with a set of smaller versions of the texture before the actual rendering takes place, thus improving both rendering performance and visual quality.
- The original texture (level zero) is downsampled to a quarter of the original area (averaging out four neighbouring texels). This downsampling is carried out recursively until one or both of the dimensions of the texture equal one texel.
- This set of images is referred to as a mipmap chain.

Mipmapping Example (from RTR)

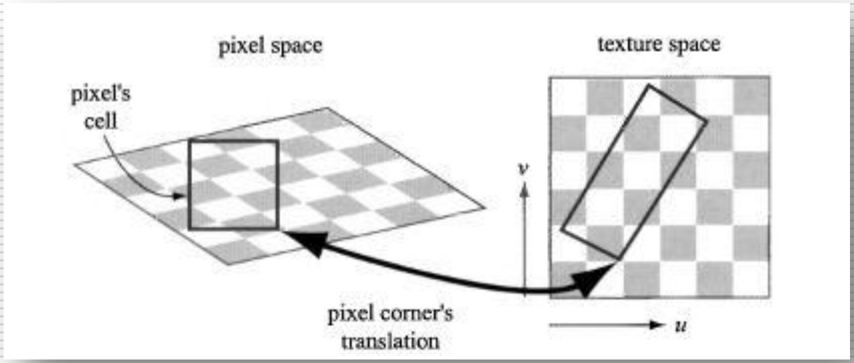


Mipmapping in OpenGL

- The texture filters available are:
  - GL_NEAREST
  - GL_LINEAR
  - GL_NEAREST_MIPMAP_NEAREST
    - Select the nearest mip level and perform nearest neighbour filtering.
  - GL_NEAREST_MIPMAP_LINEAR
    - Perform a linear interpolation between mip levels and perform nearest neighbour filter
  - GL_LINEAR_MIPMAP_NEAREST
    - Select the nearest mip level and perform linear filtering
  - GL_LINEAR_MIPMAP_LINEAR
    - Perform a linear interpolation between mip levels and perform linear filtering; also called trilinear mipmapping.

Choosing MIP level (choosing d)

- Each screen pixel encloses an area on the texture itself, as seen in the diagram below where the pixel is projected onto the texture space.



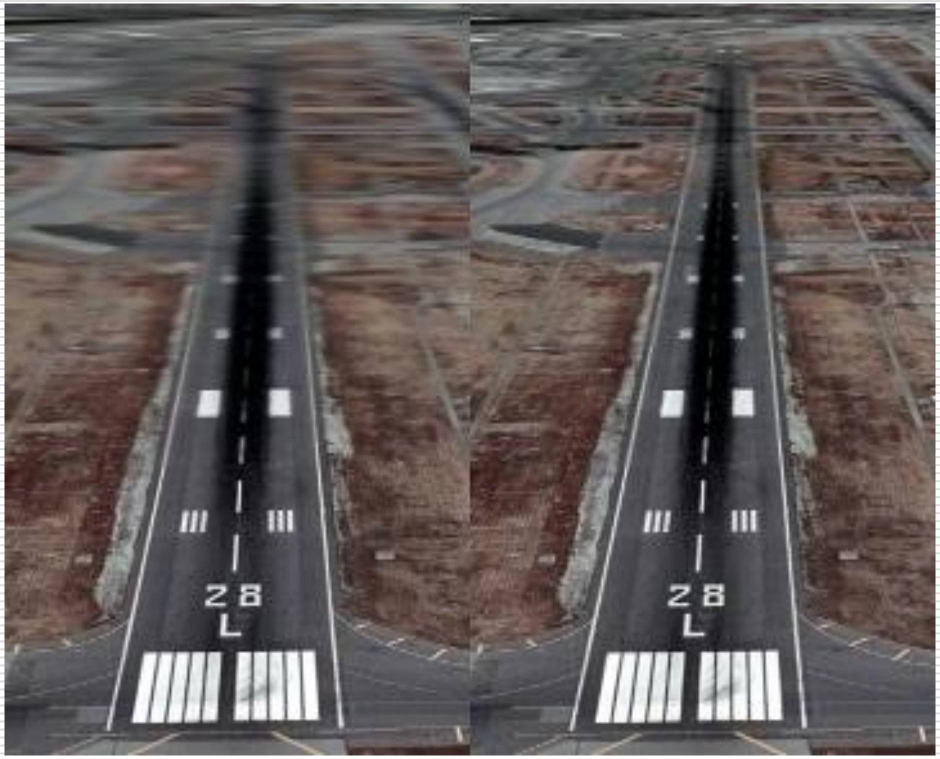
- We now need to figure out how many texels (portion of the texture) influence the pixel and from this determine the value d.

- The ideal situation would be a 1:1 ratio between pixel and texel.

- We shall not be looking at the algorithms for the best selection of d but it’s important to understand how mipmaps work.

RipMap – A Mipmap extension

- A mipmap is usually an isotropic structure, i.e. Both the x and y dimensions are the same size ... A square texture.
- However problems occur when we want to project pixels (on texels) which are oblique to the viewing angle.
- A ripmap will also sample and store textures which are rectangular.
- 1x1, 1x2, 2x1, 2x2, 2x4, etc
- ... Expensive for memory



Summed Area Table (i)

- Another way of calculating the color of a pixel (and avoid overblurring) whilst texturing is to use the SAT method.
- An array the size of the texture is created and populated (at each texel therefore) with the sum of all the corresponding texture's texels in the rectangle formed by this location and texel at the origin i.e. (0,0).
- During texturing, the pixel cell's projection onto the texture is bound by a rectanlge as seen below.

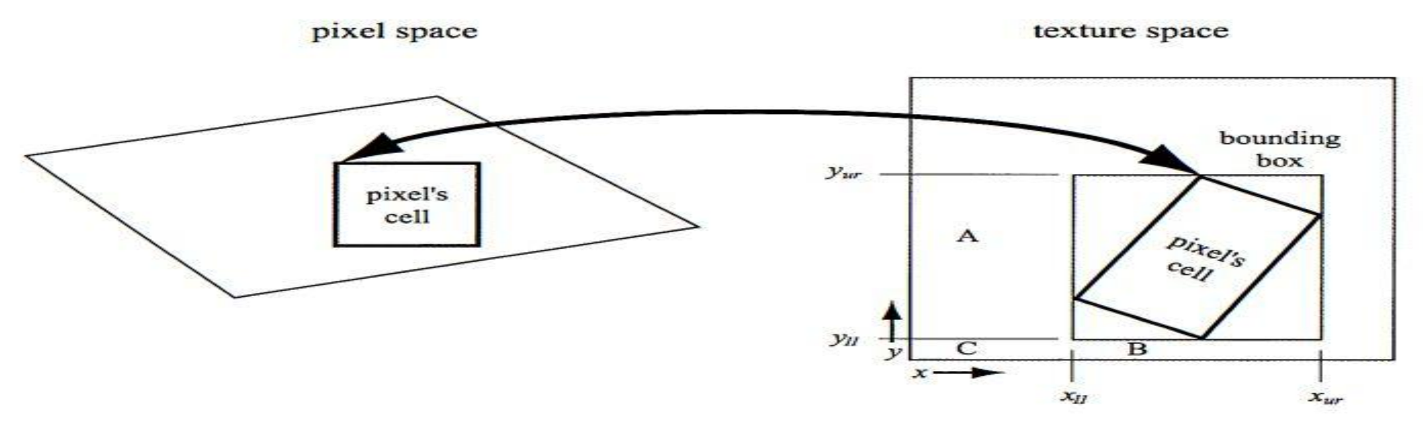


Figure 6.16. The pixel cell is back-projected onto the texture, bound by a rectangle; the four corners of the rectangle are used to access the summed-area table.

Summed Area Table (ii)

- The summed area table is then used to calculate the average colour of this rectangle, which is passed back as the texture's colour of the pixel.
- The average is calucated using texture coordinates by using the following formula:

$$\boldsymbol{c} = \frac{s[x_{ur}, y_{ur}] - s[x_{ur}, y_{ll}] - s[x_{ll}, y_{ur}] + s[x_{ll}, y_{ll}]}{(x_{ur} - x_{ll})(y_{ur} - y_{ll})}$$

- Essentially from the big box we subtract the two smaller boxes and re-add the portion (box) that was subtracted twice. We then divide by the number of texels in the bounding box.
- This method does not work very well when the back projection falls diagonally in the texture space.