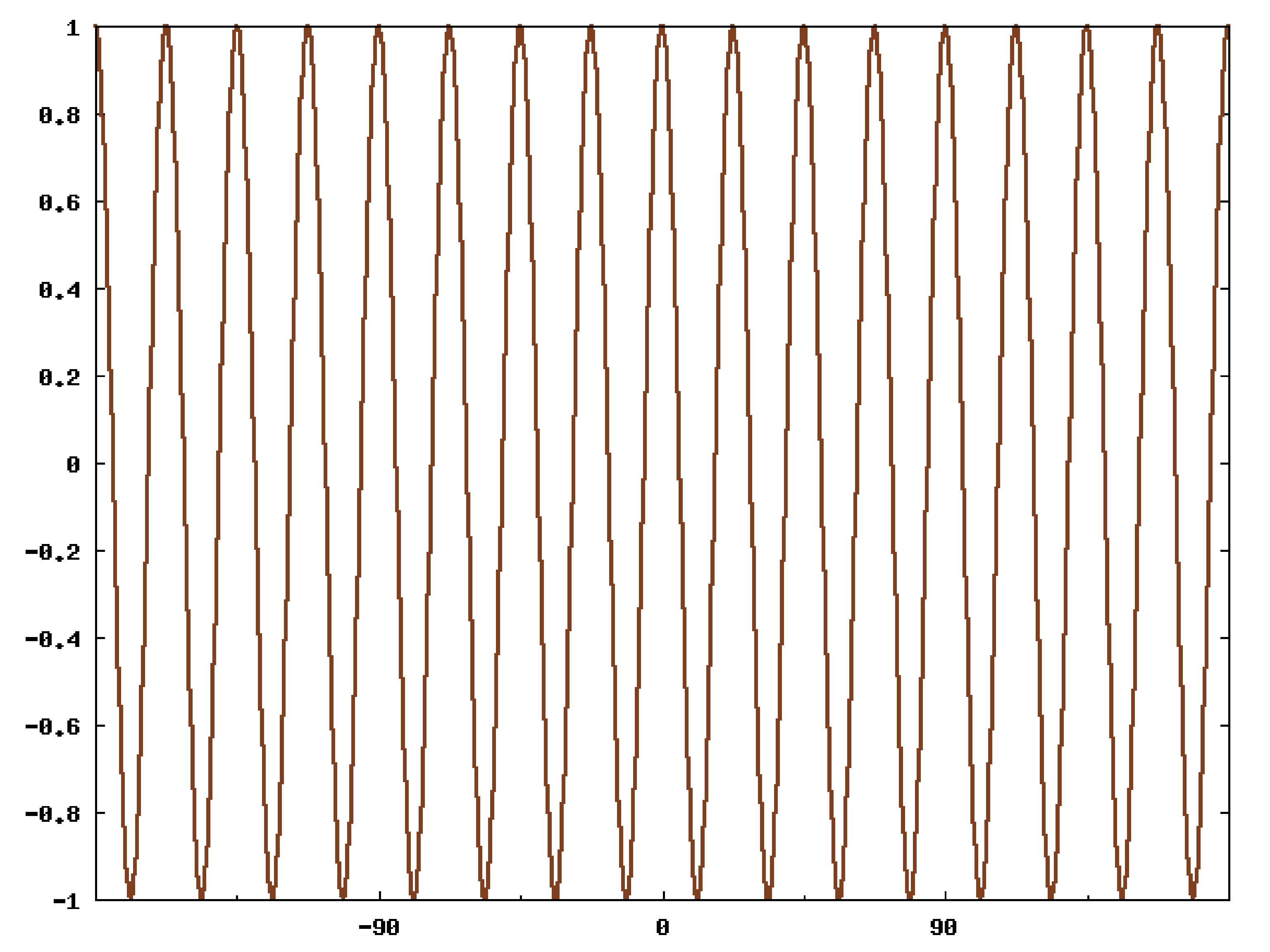
VGP351 – Week 8.1

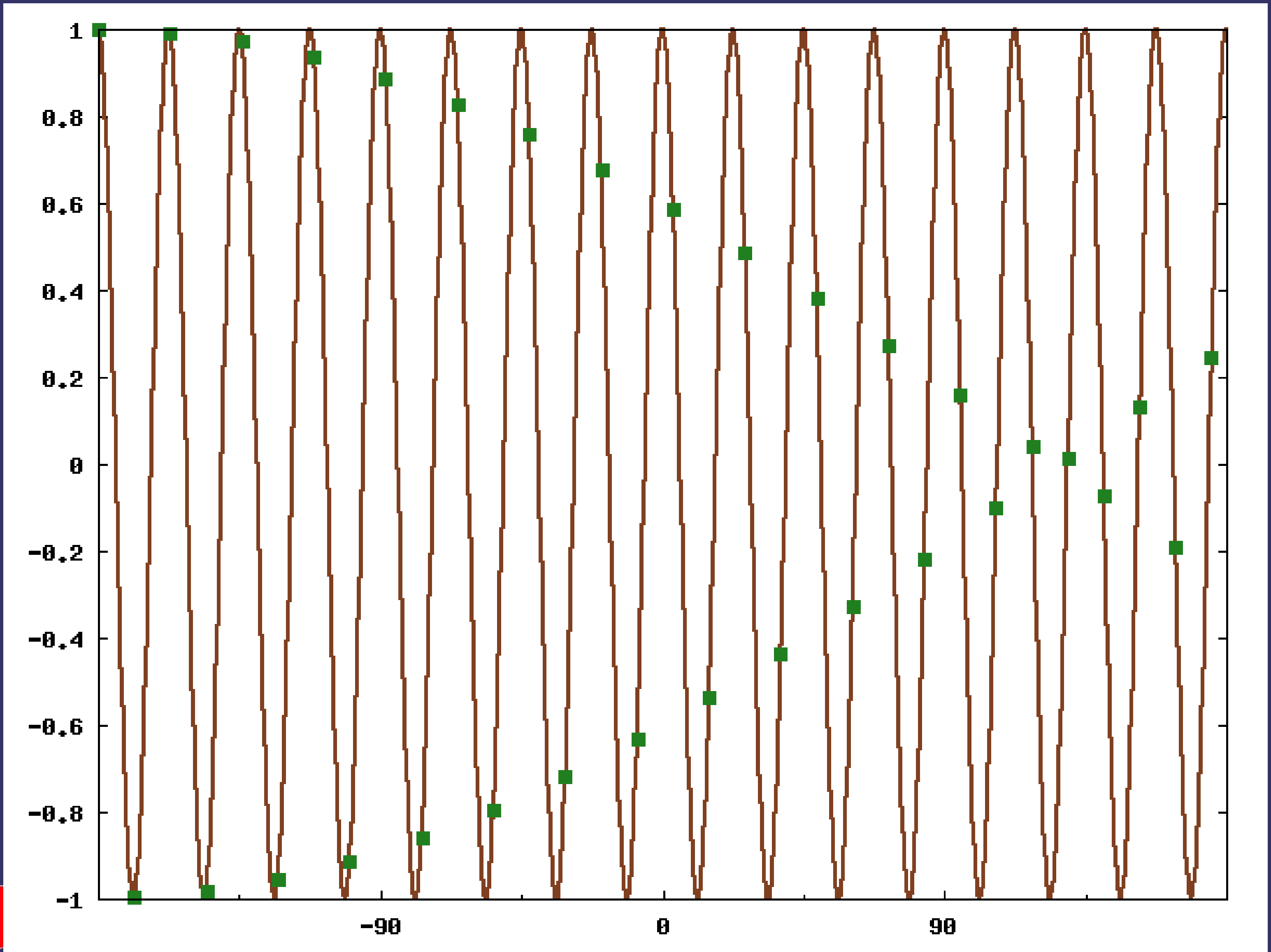
- Agenda:
  - Quiz #3
  - Sampling
    - Theory
    - Application to texture mapping
      - Simple filtering
      - Mipmapping
      - Anisotropic filtering

Sampling



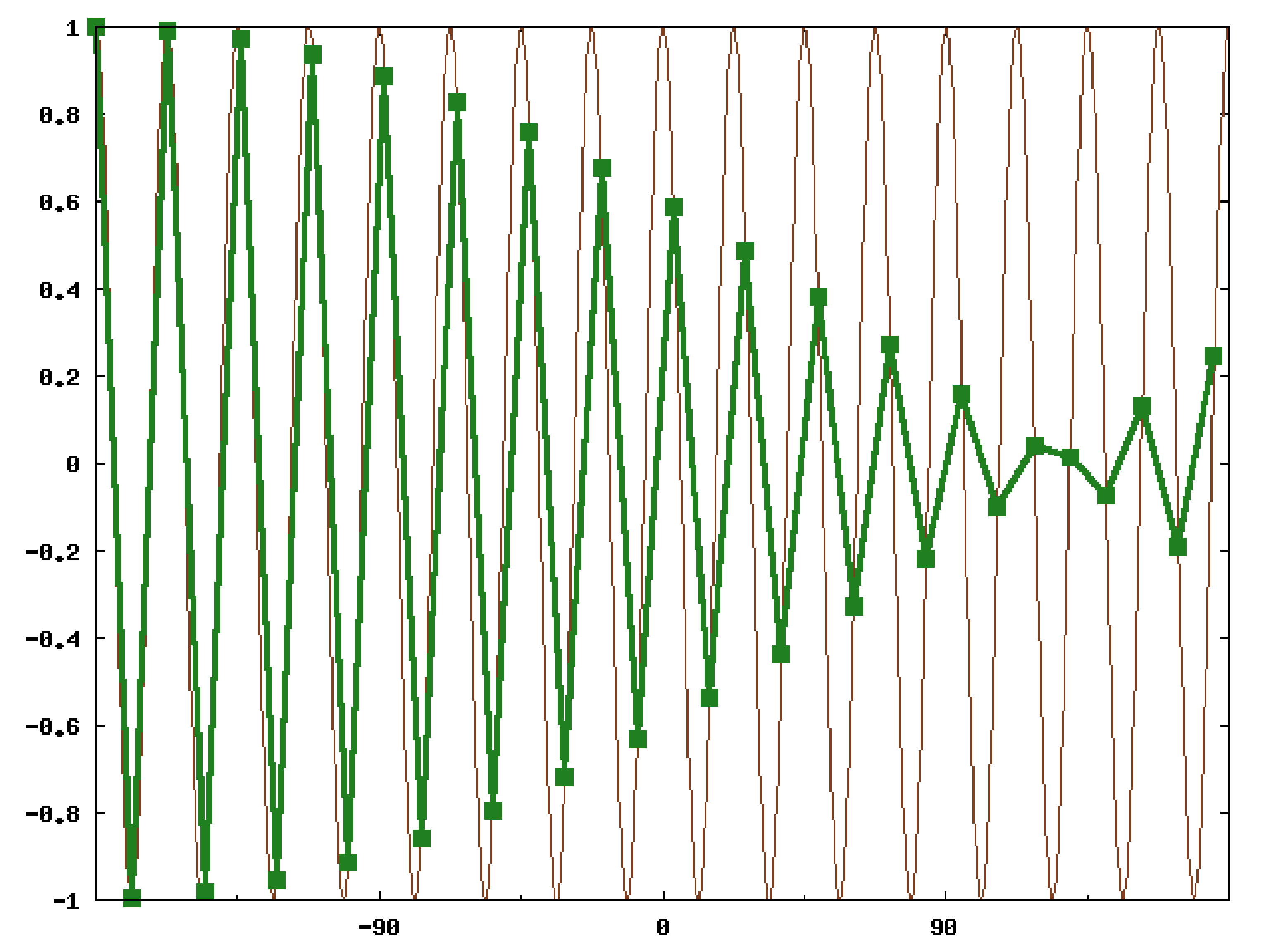
25-May-2010

Sampling

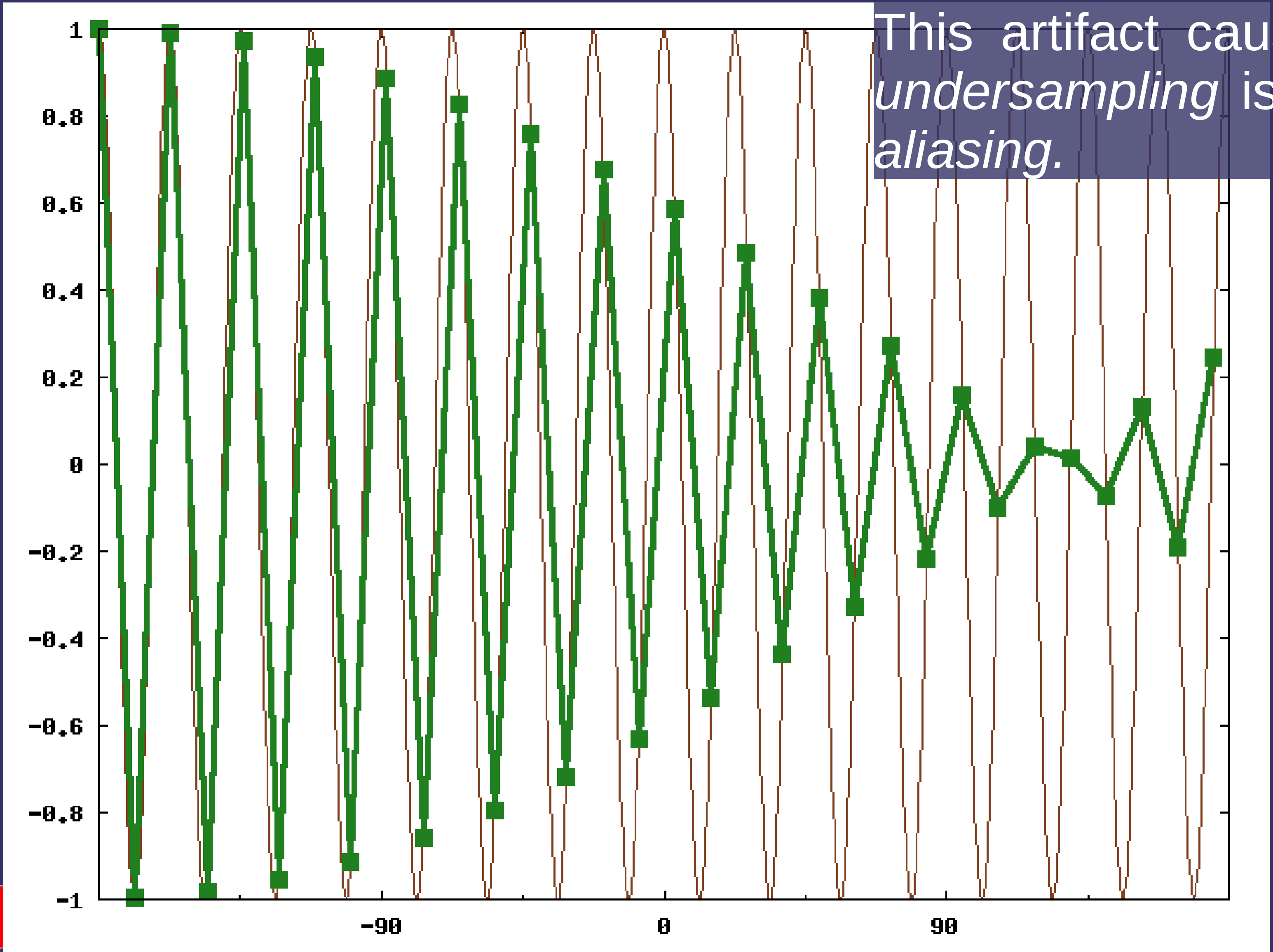


25-May-2010

Sampling



Sampling

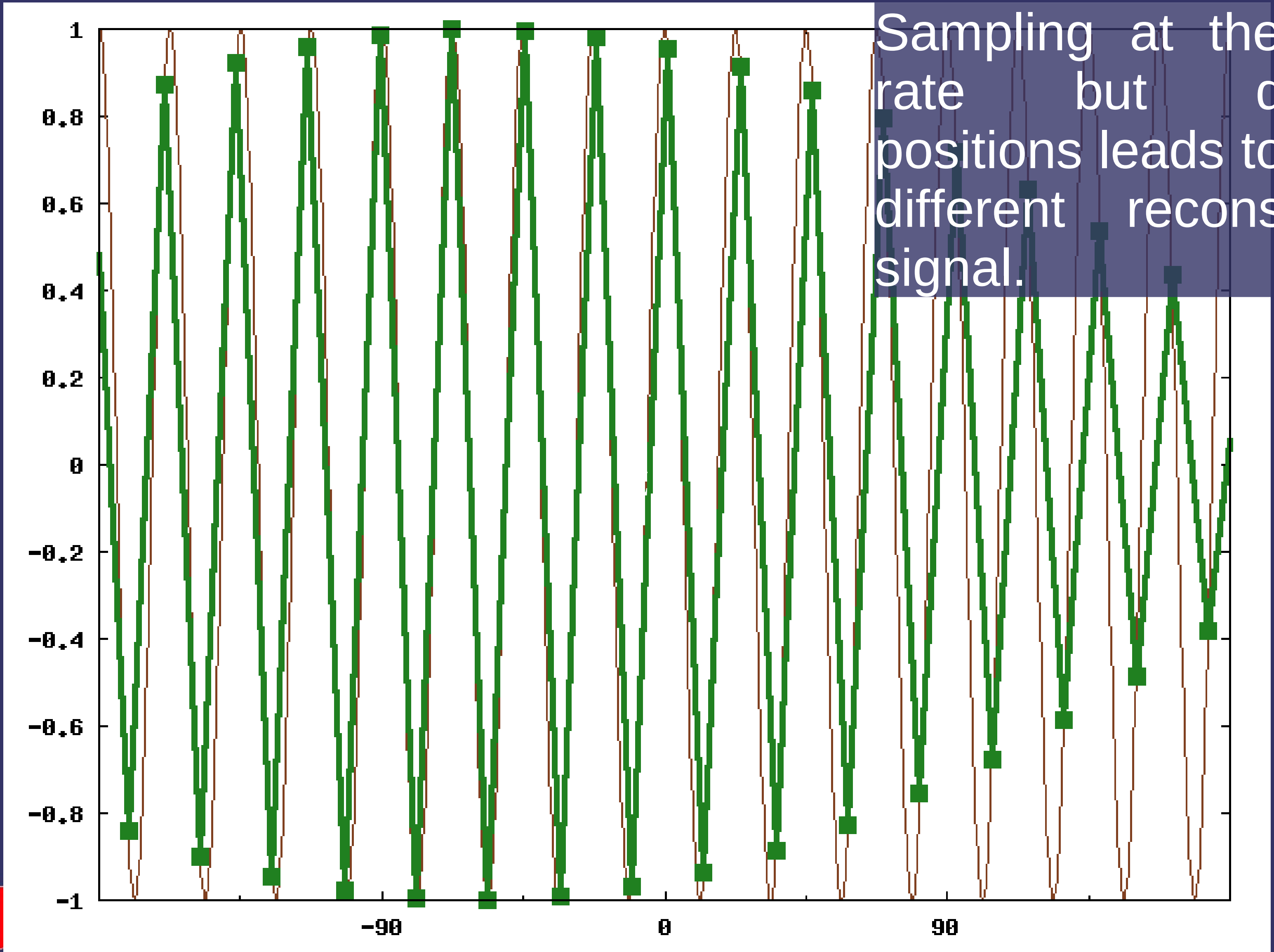


This artifact caused by *undersampling* is called *aliasing.*

25-May-2010

© Copyright Ian D. Romanick 2009, 2010

Sampling



Sampling at the same rate but different positions leads to a very different reconstructed signal.

Avoiding Aliasing

- How?

Avoiding Aliasing

- How?
- Sample at a higher rate
  - What sample rate is sufficient?
  - More samples means more data, and that comes at a cost

Nyquist-Shannon Sampling Theorem

- If f is the highest frequency element in a signal, the signal must be sampled at a rate of at least $2f$ in order to be accurately reconstructed
  - If the sample rate is f_s then we call $f_s/2$ the *critical frequency* or the *Nyquist frequency*
  - Any elements in the signal with frequency higher than the critical frequency will alias

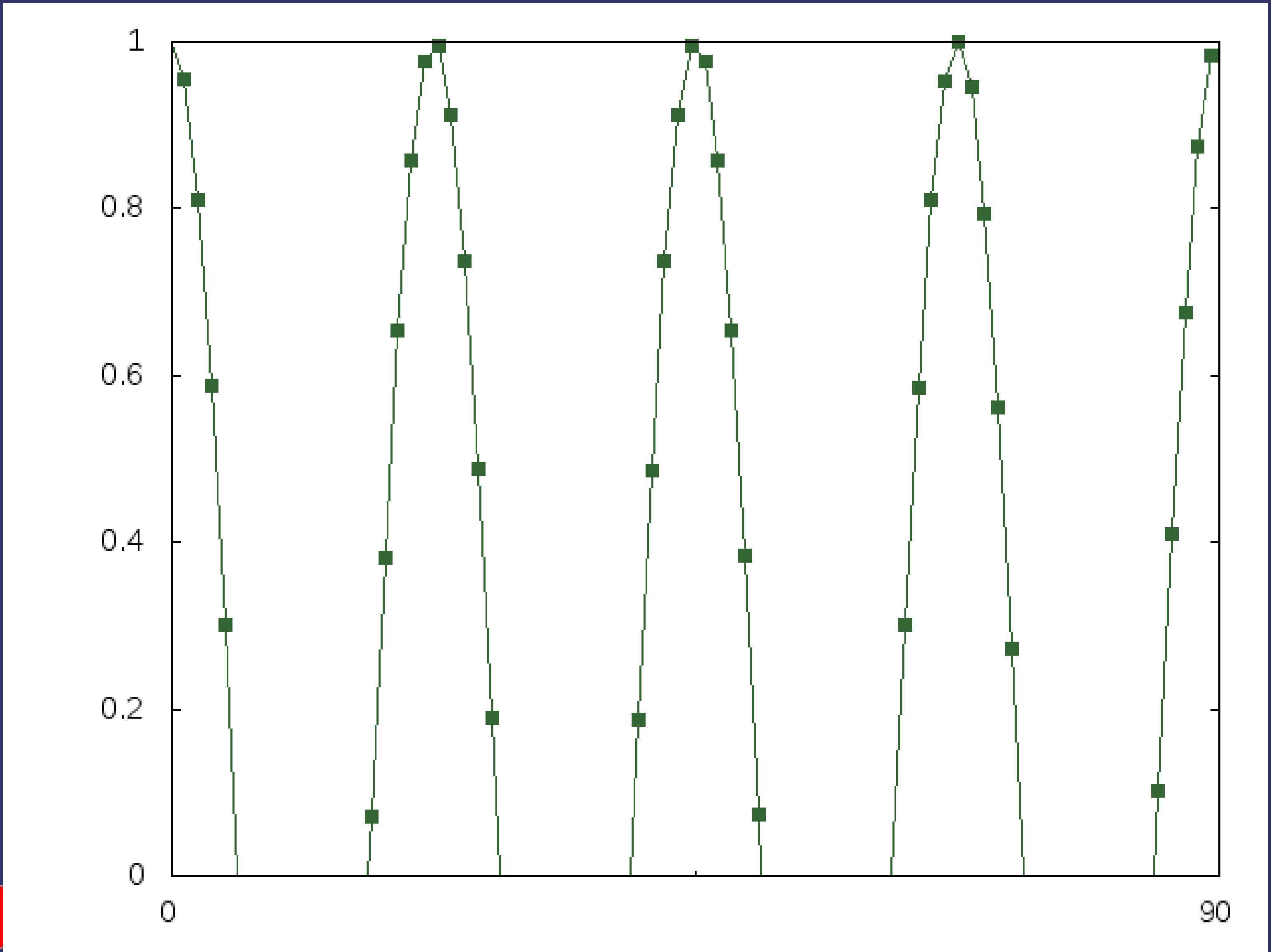
Avoiding Aliasing

- If having frequencies above the critical frequency causes aliasing, how can we eliminate the aliasing?

Avoiding Aliasing

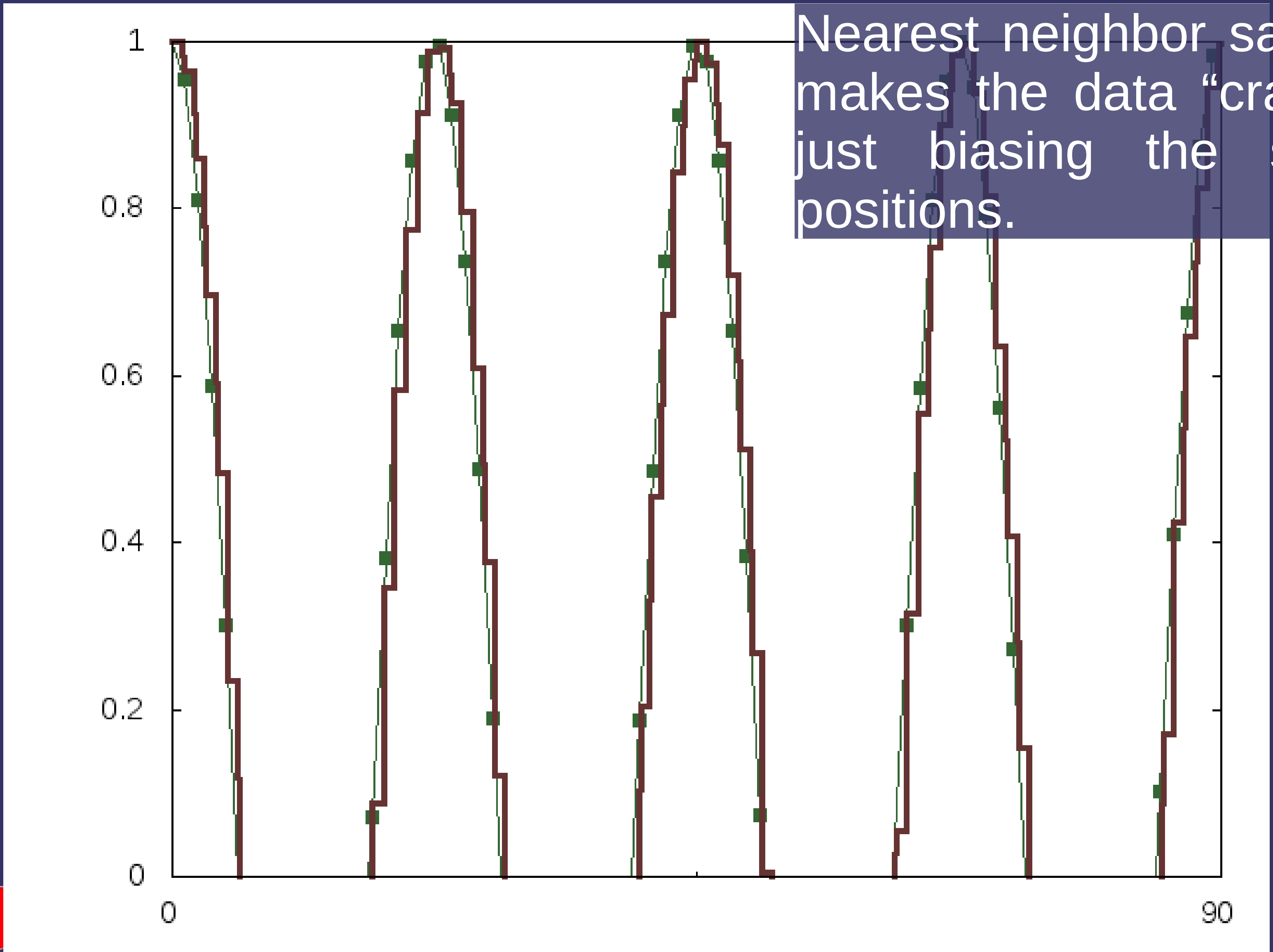
- If having frequencies above the critical frequency causes aliasing, how can we eliminate the aliasing?
  - Remove elements above the critical frequency!
  - This is done using a low-pass filter

Resampling



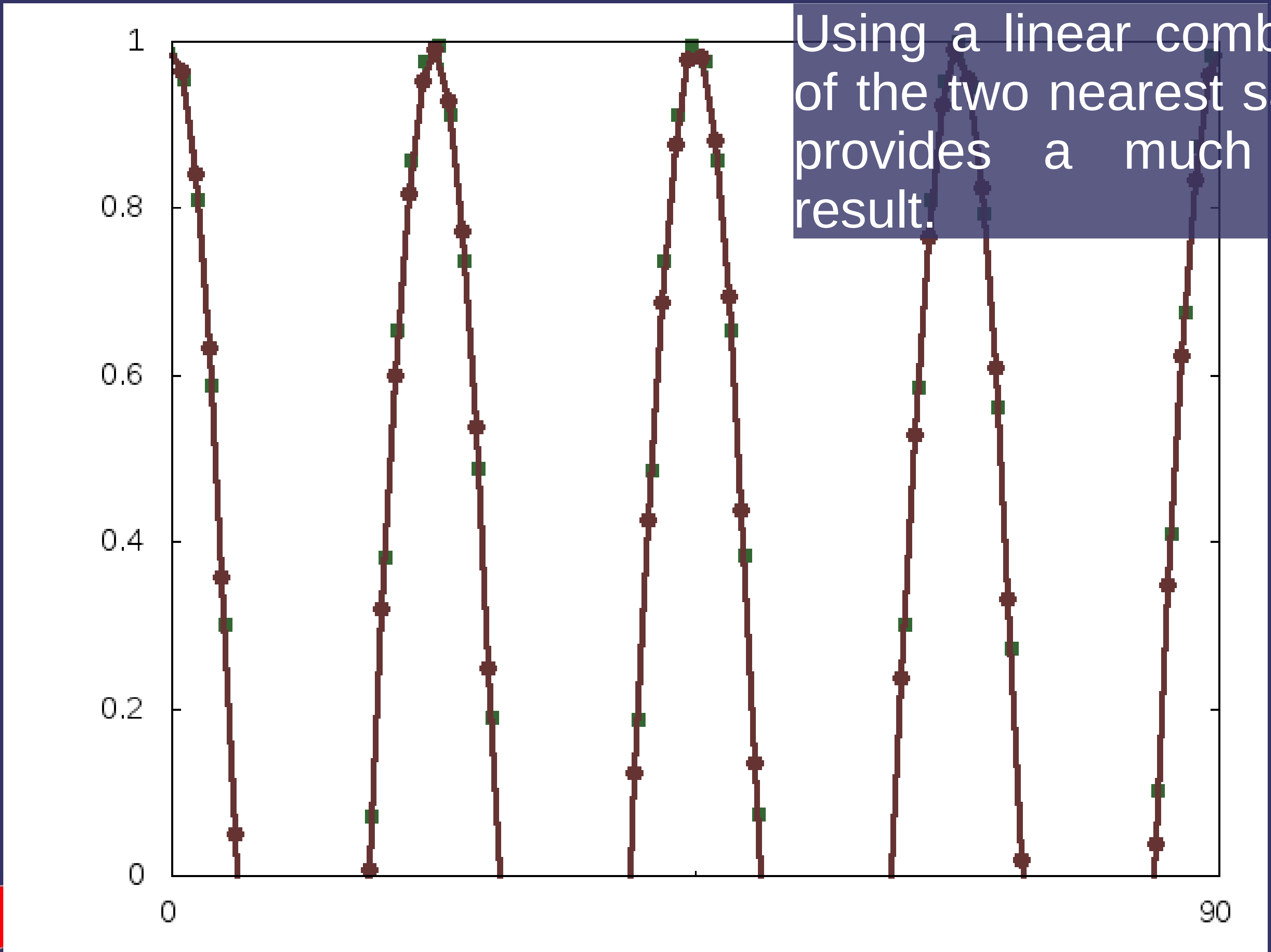
25-May-2010

Resampling



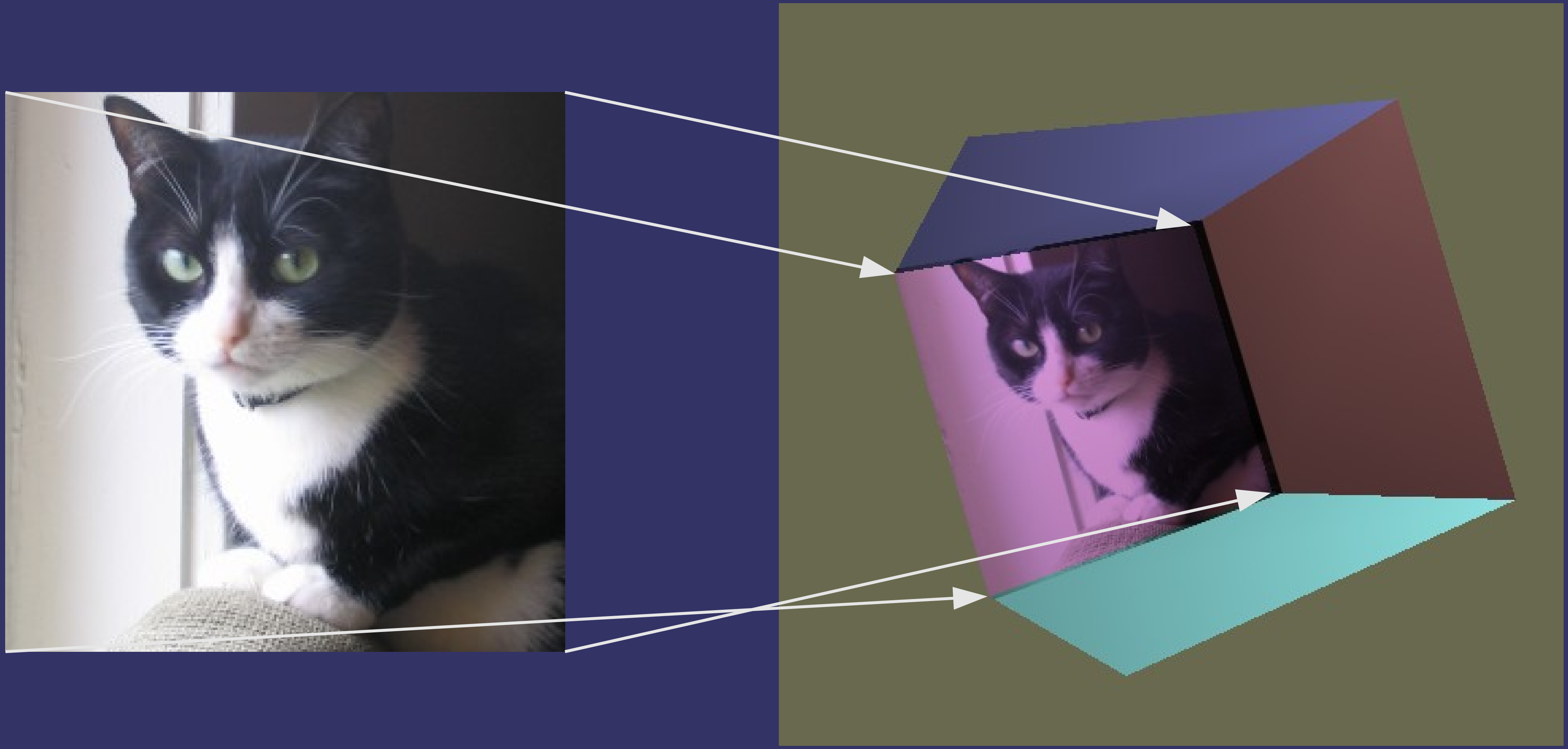
Nearest neighbor sampling makes the data “crawl” by just biasing the sample positions.

Resampling



Using a linear combination of the two nearest samples provides a much better result.

Texture Mapping



Magnification

- When a single texel is mapped to multiple fragments, the texture is magnified
- What happens when the location sampled from the texture lies between texels?

Magnification

- When a single texel is mapped to multiple fragments, the texture is magnified
- What happens when the location sampled from the texture lies between texels?
  - Nearest neighbor sample
  - Linear sample
  - Cubic convolution
    - Rarely implemented in hardware, but you could write a shader to do it!

Minification

- When a single fragment covers multiple texels, the texture is minimized
  - This is where texture aliasing can occur
- What to do?

Minification

- When a single fragment covers multiple texels, the texture is minimized
  - This is where texture aliasing can occur
- What to do?
  - In a perfect world, sample and filter all of the covered texels
  - Since an entire 1024×1024 texture could be minimized to a single fragment, this is impractical

Minification

- Nearest neighbor sampling
  - Most likely to have aliasing

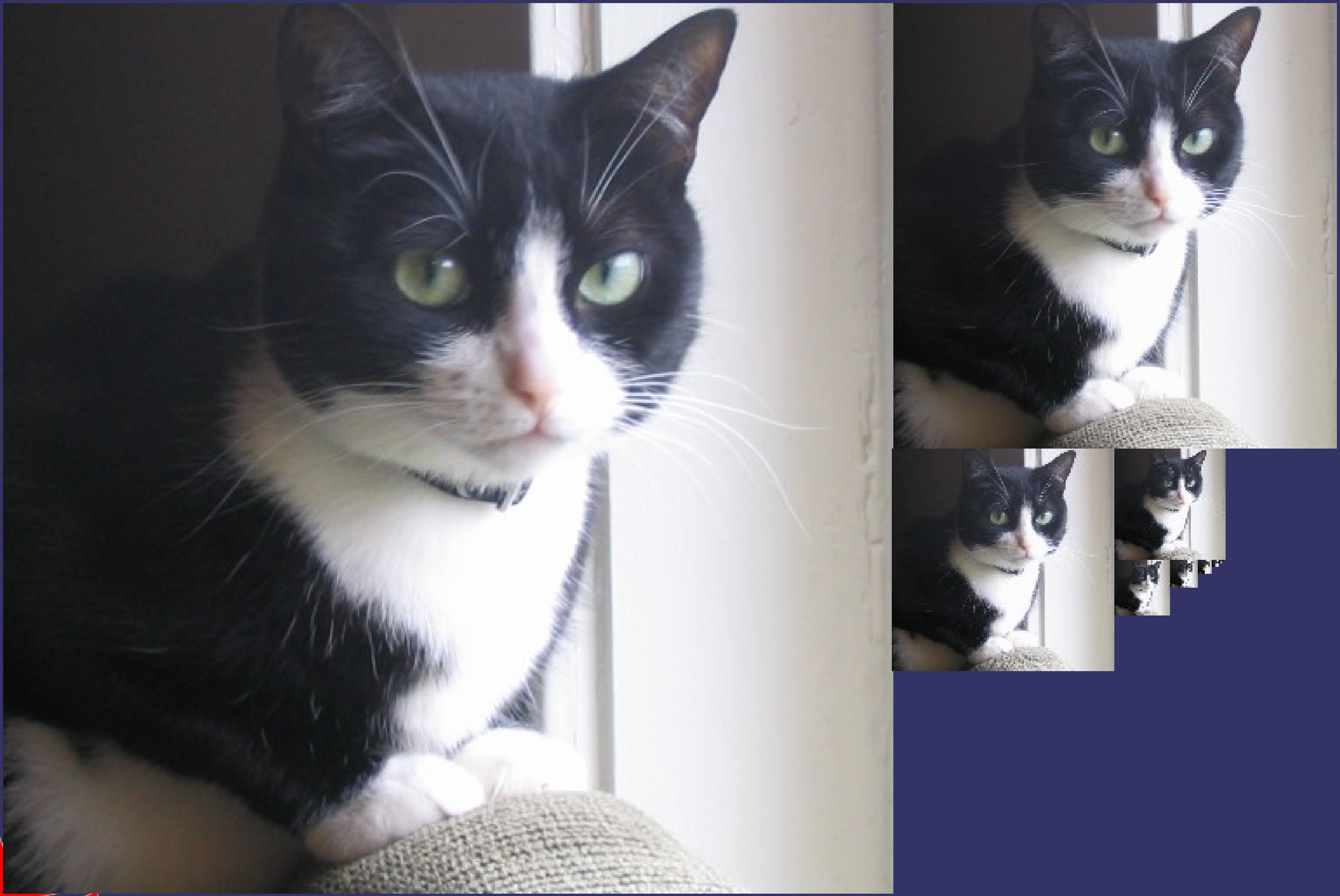
Minification

- Linear filtering of nearest neighbors
  - In 2D this is called *bilinear filtering*
  - Better results because we're effectively doubling our sample rate
    - We also increase the memory bandwidth requirements by 2^n
  - At some point the texture will be minimized enough that the sample rate will still be too low to prevent aliasing

Mipmapping

- Create multiple pre-filtered, down-sampled versions of the "base" texture
  - Down-sampled textures are called *mipmaps*
  - The collection of mipmaps for a particular base texture is called its *mipmap stack*
  - From Latin "multum in pavro" for "many things in one place"
- As the texel area covered by a fragment increases, use a smaller mipmap
  - In smaller mipmaps, each texel represents more samples from the base texture

Example Mipmap Stack



Mipmapping

- What's the trade-off?

Mipmapping

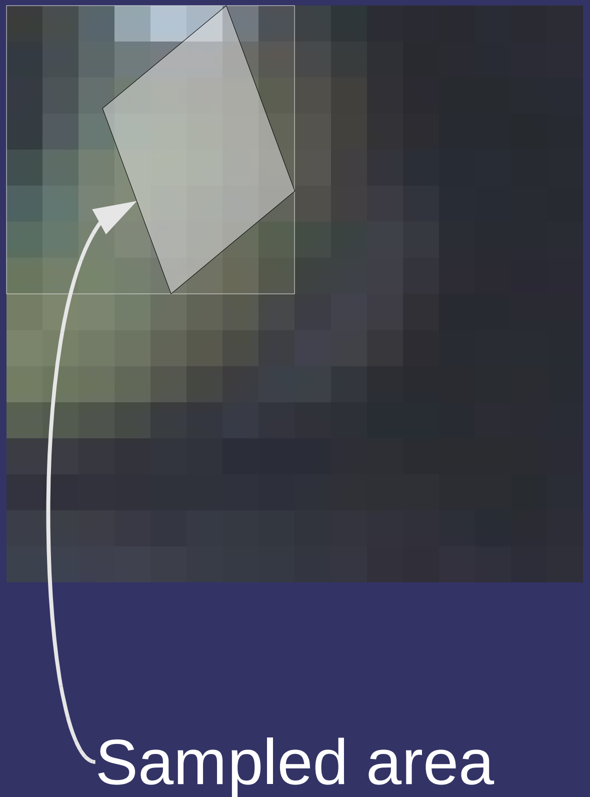
- What's the trade-off?
  - Memory *size* versus memory *bandwidth*
  - What is the increase in size for a 2D texture?

Mipmapping

- What's the trade-off?
  - Memory *size* versus memory *bandwidth*
  - What is the increase in size for a 2D texture?

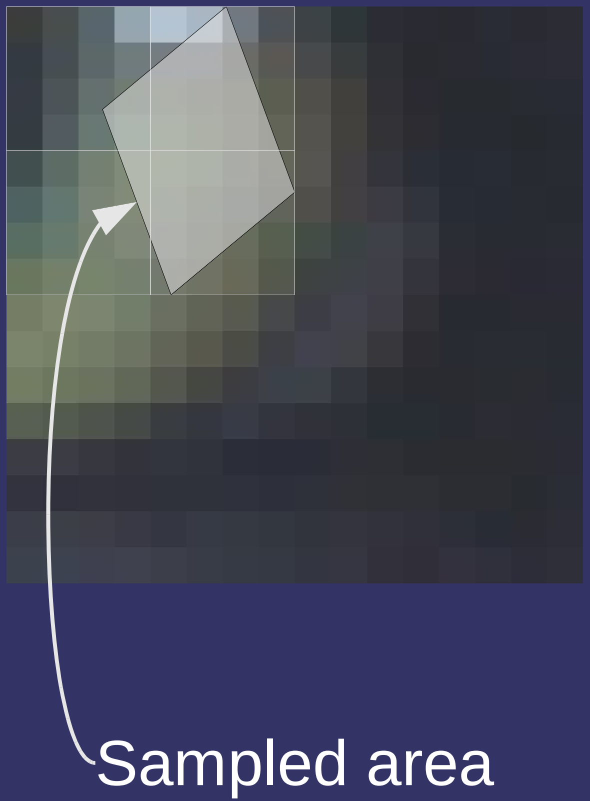
$$\frac{1}{2^2}+\frac{1}{4^2}+\frac{1}{8^2}+\ldots+\frac{1}{2^{2n}}\approx\frac{1}{3}$$

Mipmapping



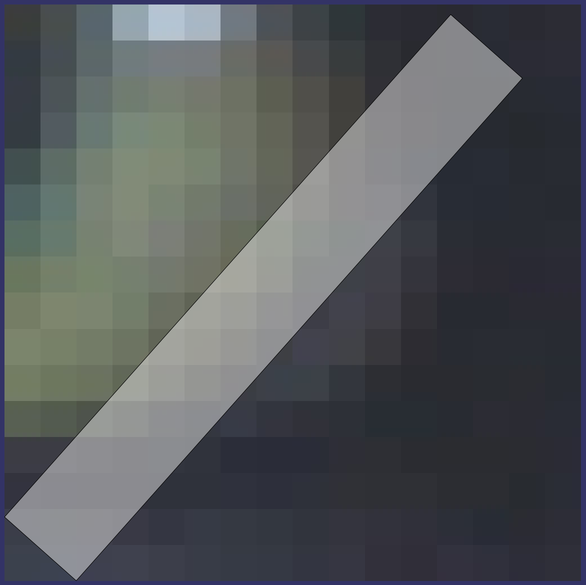
- LOD will be used where the outlined area is a single texel
  - No aliasing, but lots of unneeded data is filtered in
  - Results in images that are too blurry or over-filtered

Mipmapping



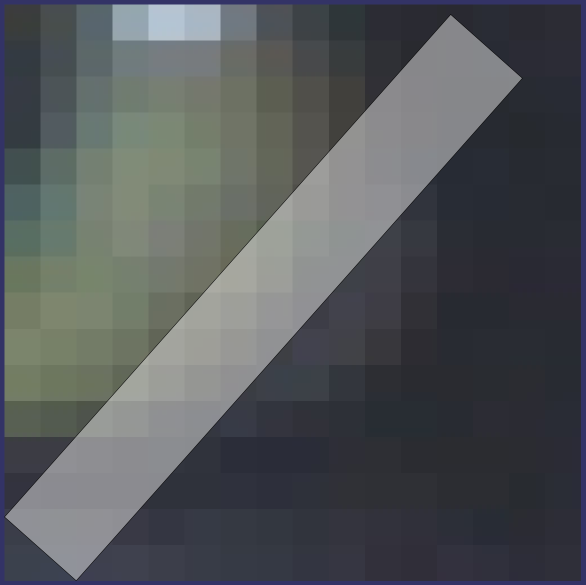
- Can *partially* fix the oversampling by taking multiple samples from the next higher LOD
  - This is a bi-linear filter of the mipmap
  - Can extend further by filtering between LODs

Mipmapping



- For this case, mipmap filtering will either oversample or undersample

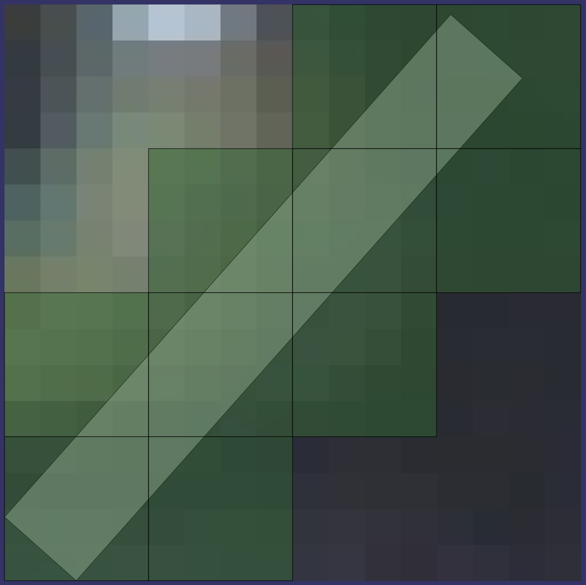
Improved Filtering



- All of these filter modes assume that the sample region is *isotropic*
  - Isotropy is the property of being uniform in all directions
  - We clearly can have ideal sample regions that are *anisotropic*

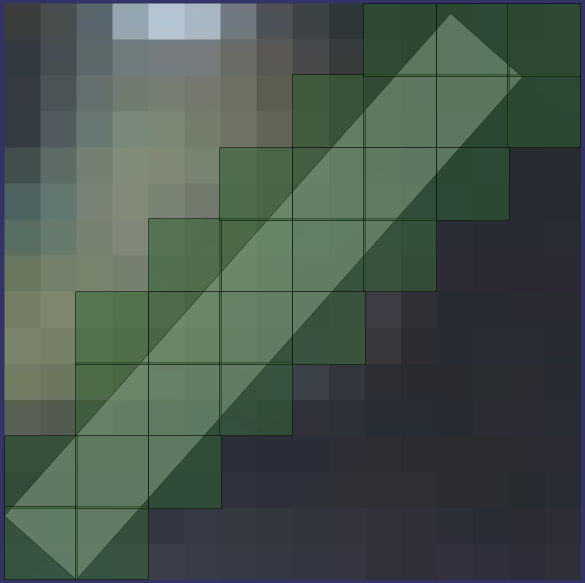
25-May-2010

Improved Filtering



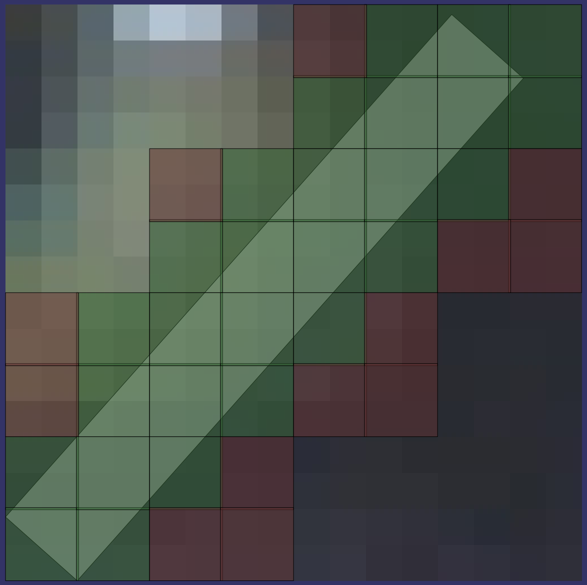
- An anisotropic filter might sample these 10 positions in the appropriate mipmap

Improved Filtering



- An anisotropic filter might sample these 27 positions in the appropriate mipmap

Improved Filtering



- An anisotropic filter might sample these 27 positions in the appropriate mipmap
  - The red boxes show the regions where over-filtering would occur with only 10 samples

Setting Filter Modes

- OpenGL has a name for each each of these filter modes:
  - `GL_NEAREST` – Point sampling
  - `GL_LINEAR` – Bi-linear in 2D
  - `GL_NEAREST_MIPMAP_NEAREST` – Point-sampling from mipmap
  - `GL_LINEAR_MIPMAP_NEAREST` – Linear sampling from one mipmap
  - `GL_NEAREST_MIPMAP_LINEAR` – Linear blend of two point-sampled mipmaps
  - `GL_LINEAR_MIPMAP_LINEAR` – Linear blend of two bi-linear sampled mipmaps. Also known as *tri-linear filtering* in 2D

Setting Filter Modes

- Set texture filter modes with:

```
void glTexParameteri(GLenum target,
    GLenum pname, GLint param);
```

  - `pname` is either `GL_TEXTURE_MAG_FILTER` or `GL_TEXTURE_MIN_FILTER`
  - `param` is one of the modes from the previous page

Setting Filter Modes

- Texture filter anisotropy is controlled by setting `GL_TEXTURE_MAX_ANISOTROPY_EXT`
  - Maximum amount of anisotropy is queried by `GL_MAX_TEXTURE_MAX_ANISOTROPY_EXT` to `glGetIntegerv`
  - Requires that the extension `GL_EXT_texture_filter_anisotropic` be available

Setting Mipmaps

- Mipmap is selected with the level parameter to the `glTexImage` functions:

```
void glTexImage1D(GLenum target, GLint level,
    GLint internalFormat, GLsizei width,
    GLint border, GLenum format, GLenum type,
    const GLvoid *pixels);
```

  - Zero is the “base” level, 1 is ½ size, 2 is ¼ size, etc.
  - Textures that use mipmap filtering must be *mipmap complete*
    - All mipmaps down to 1×1 that might be used must be specified

Mipmap Generation

- OpenGL can automatically generate the full set of mipmaps each time the base level is modified
  - Set `GL_GENERATE_MIPMAP` to `GL_TRUE`
  - This causes the mipmap stack to be regenerated if even one texel is modified in the base level!

Mipmap Generation

- Later versions of GL add a different mechanism

```
void GenerateMipmapEXT(GLenum target);
```

  - Generates mipmaps from base level to max level
  - Function only available if `GL_EXT_framebuffer_object` is supported
    - Drop “`EXT`” from the name if OpenGL 3.0 or `GL_ARB_framebuffer_object` is supported

LoD Clamping

- Used mipmaps can be restricted to a subset of the possible range
  - `GL_TEXTURE_BASE_LEVEL` specifies the base level. The default is zero.
  - `GL_TEXTURE_MAX_LEVEL` specifies the highest level (smallest mipmap / lowest LoD) that will be used.
  - These settings also affect automatic mipmap generation

Next week...

- Texture mapping part 3
  - Environment mapping
  - Projective texturing
  - Texture atlases
  - Texture compression

Legal Statement

This work represents the view of the authors and does not necessarily represent the view of Intel or the Art Institute of Portland.

OpenGL is a trademark of Silicon Graphics, Inc. in the United States, other countries, or both.

Khronos and OpenGL ES are trademarks of the Khronos Group.

Other company, product, and service names may be trademarks or service marks of others.