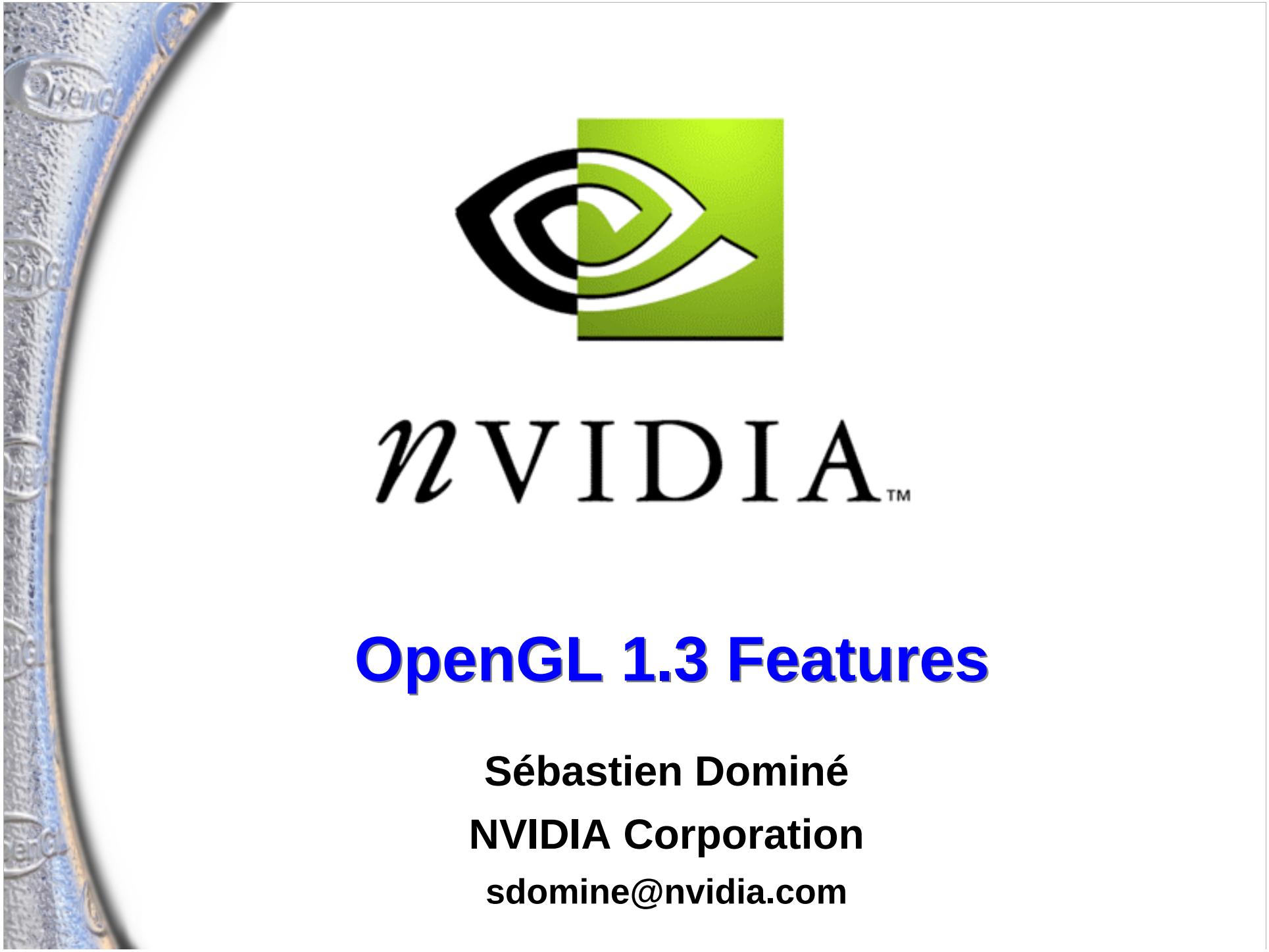


OpenGL 1.3 Features

Sébastien Dominé

NVIDIA Corporation

sdomine@nvidia.com

Agenda

- Introduction
- OpenGL 1.3 feature set
  - Compressed Textures
  - Cube Map Textures
  - Multisample
  - Multitexture
  - Texture Add Environment Mode
  - Texture Combine Environment Mode
  - Texture Dot3 Environment Mode
  - Texture Border Clamp
  - Transpose Matrix
- Perspectives

Introduction

- The ARB (Architecture Review Board) promotes standardized extensions (_ARB) to be part of the GL core
- OpenGL 1.2
  - 3D textures
  - BGRA
  - Packed Pixel Formats
  - Normal Rescaling
  - Separate Specular Color
  - Texture Coordinate Edge Clamping
  - Texture Level Of Detail Control
  - Vertex Array Draw Element Range
  - Imaging Subset (Color Tables, Convolution, Color Matrix, etc…)
- OpenGL 1.2.1
  - ARB extensions – GL_ARB_multitexture
- OpenGL 1.3

GL Extensions

- **Recipe:**
  - **Get the extensions string:**

    `glGetString(GL_EXTENSIONS)`

    - **If the app copies the extensions string, don't allocate a fixed size string, but look at the real size of the string returned by GL**
  - **Search for the sub-string**
  - **Make sure the character at the end of the sub-string is either a space or a NULL character – Avoid "XXX_extension" to be found when "XXX_extension_2" is only available**
  - **Assign the entry points as needed by calling**
    - **wglGetProcAddress(<entry point string>)**

WGL Extensions

- Windows GL Extensions
- WGL_ARB_extensions_string

wglGetProcAddress("wglGetExtensionsStringARB")

- Verify the entry point returned is non NULL
- Do the the WGL_ based extensions string search using the `wglGetExtensionsStringARB` entry point

Extensions Utilities

- **NVIDIA OGL SDK**
  - **glh_extensions.h : helper that takes care of it. The user passes a space separated list of required extensions string and if it succeeds, everything gets initialized properly**

```
int glh_init_extensions(const char *origReqExts)
```

Compressed Textures

- Based upon GL_ARB_texture_compression
- Allows the application to upload texture maps to GL in a compressed form:
  - Already Compressed pixel buffer
  - Use the driver to perform the compression
- The compressed texture upload mechanism can be algorithm independent

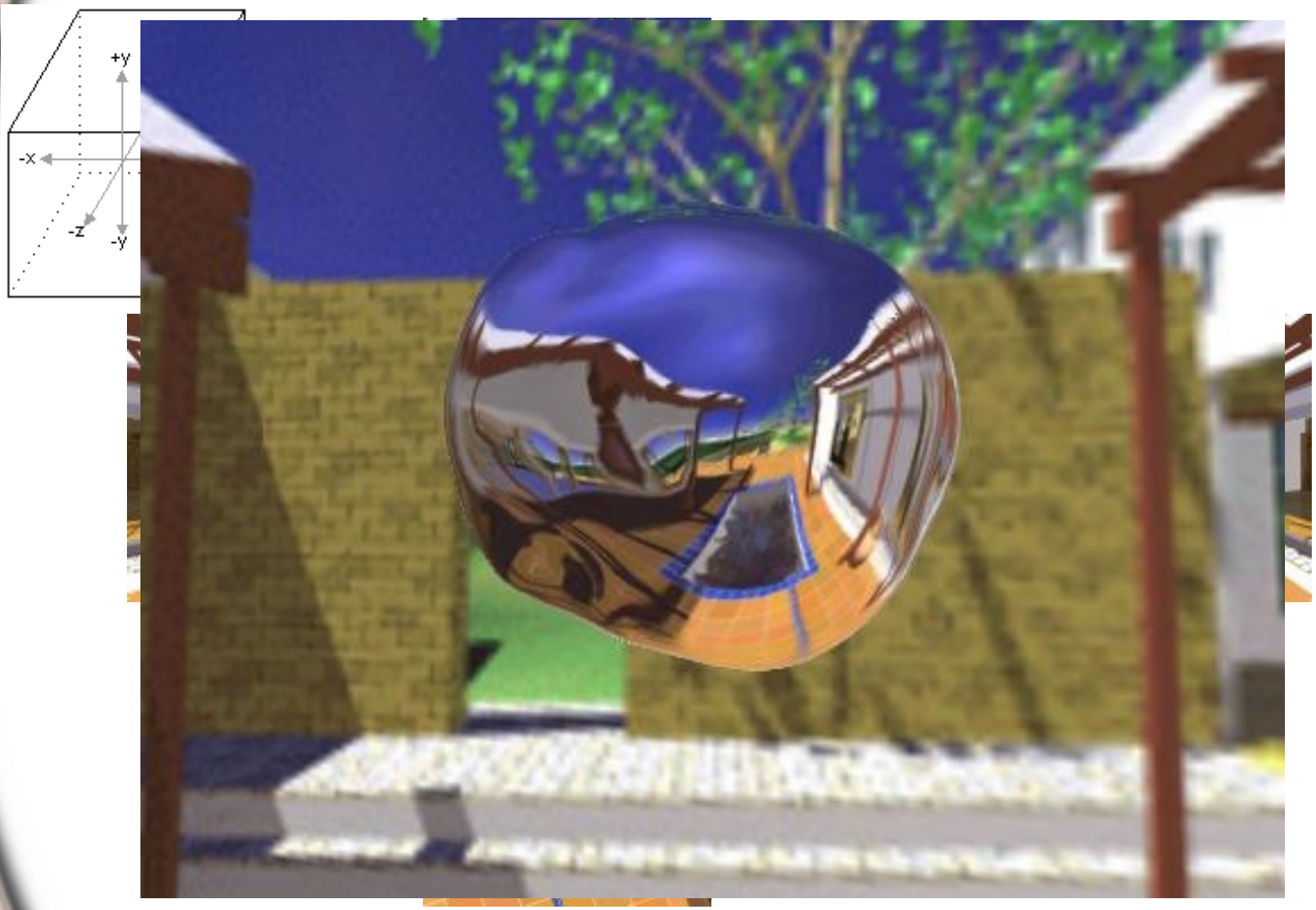
Compressed Textures

- **Benefits:**
  - **Minimizes Texture Memory Usage**
  - **Minimizes Bus Bandwidth for Texture Uploads**
  - **Faster Texture Uploads – Compressed Textures on Disk**
  - **Texture Cache Friendly – improves rasterization performance**
- **Disadvantages:**
  - **Image Quality may suffer – be careful**
  - **Slower to have the driver do the compression**

Cube Map Textures

- Based upon GL_ARB_texture_cube_map
- A Cube Map Texture is defined by a set of 6 2D images that represents the 6 faces of a cube
- Texture coordinates (s,t,r) fetch the texture by using these coordinates as a direction vector emanating from the center of a cube
- Left Handed System (RenderMan, DX7)
- Use GL_CLAMP_TO_EDGE clamp mode for seamless junctions
- New Texture Generation modes:
  - Eye-space vertex's normal GL_NORMAL_MAP_ARB
  - Eye-space vertex's reflection vector GL_REFLECTION_MAP_ARB

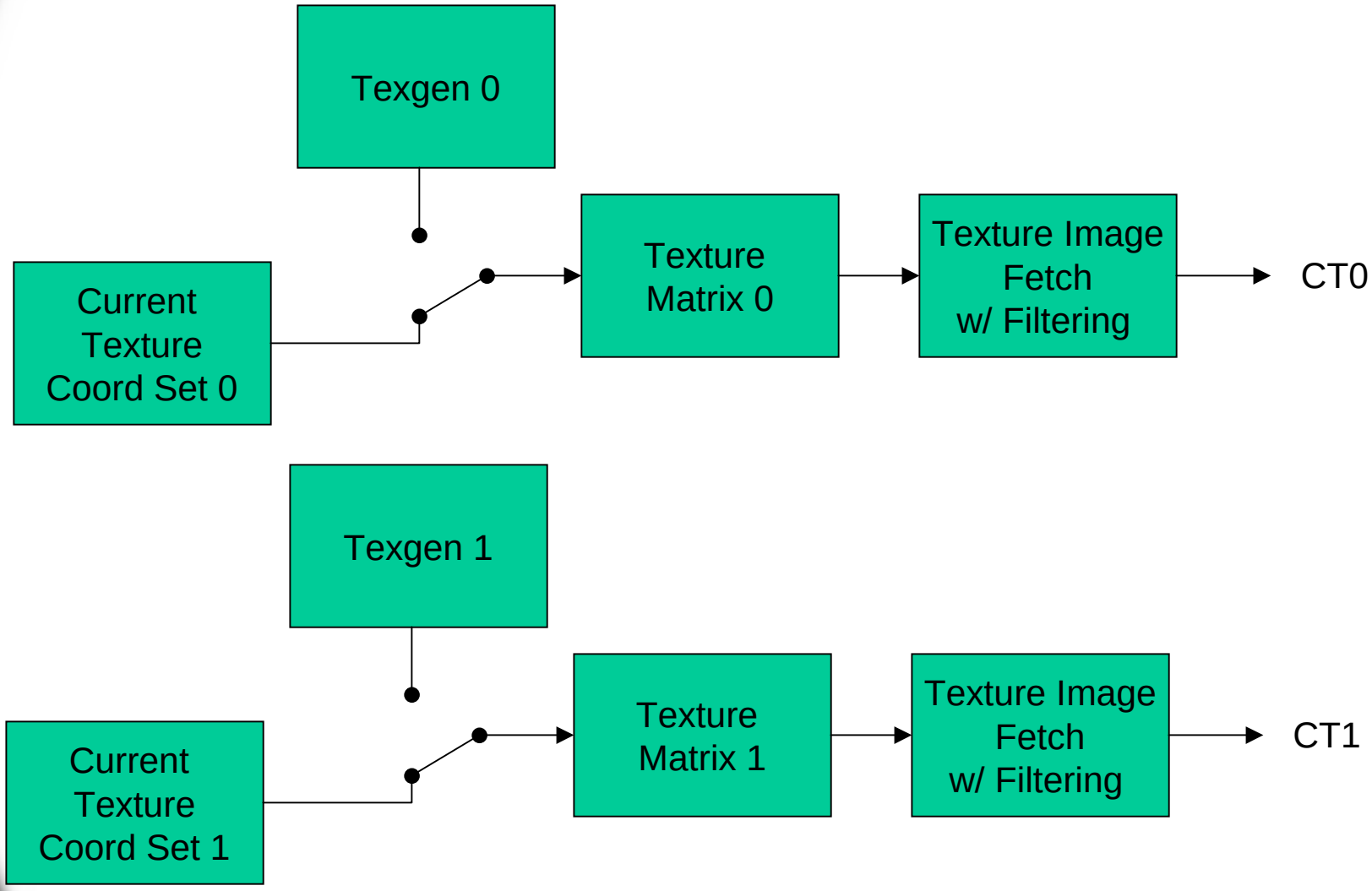
Cube Map Textures



Multitexture

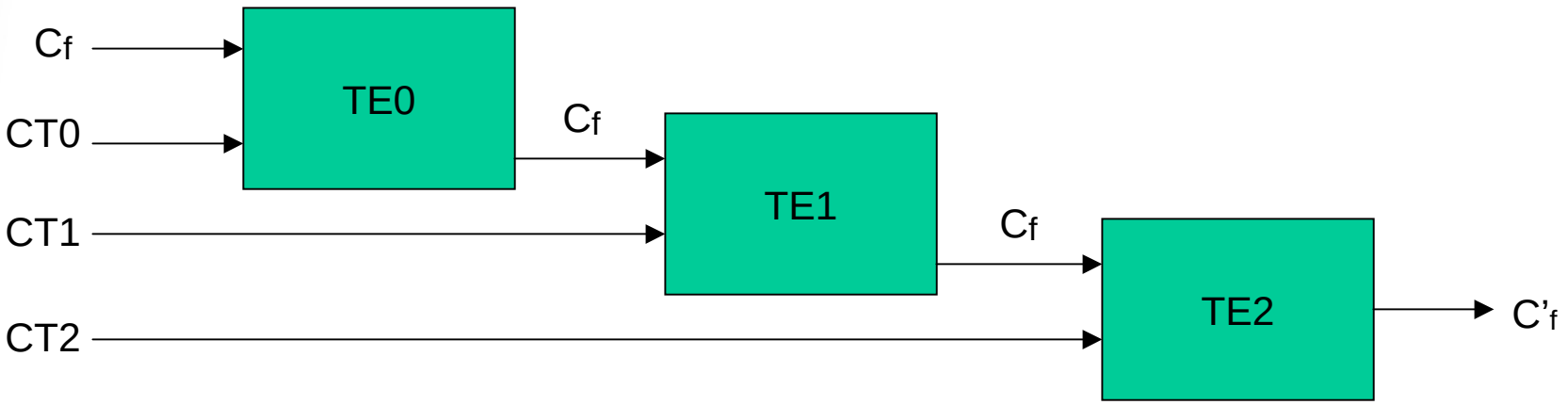
- Based upon GL_ARB_multitexture
- Multiple texture units that have their own:
  - Set of Texture Coordinates
  - Texture image w/ Filtering parameters
  - Texture Environment functions (REPLACE, MODULATE, DECAL, BLEND)
  - Texture Generation mode
  - Texture Matrix

Multitexture



Multitexture

- **Texture Environment**



C_f = Fragment color input to texturing

CT_i = Texture color fetched and filtered texture unit i

TE_i = Texture environment i

C'_f = Fragment color output from texturing

Multitexture

- **Texture Environment Modes**

Base internal format	REPLACE	MODULATE	DECAL	BLEND
RGBA (or 4)	$R=R_t$ $G=G_t$ $B=B_t$ $A=A_t$	$R=R_f R_t$ $G=G_f G_t$ $B=B_f B_t$ $A=A_f A_t$	$R=R_f (1-A_t)+R_t A_t$ $G=G_f (1-A_t)+G_t A_t$ $B=B_f (1-A_t)+B_t A_t$ $A=A_f$	$R=R_f (1-R_t)+R_c R_t$ $G=G_f (1-G_t)+G_c G_t$ $B=B_f (1-B_t)+B_c B_t$ $A=A_f A_t$

Texture Add Environment Mode

- **New Texture Environment function (ADD)**

Base internal format	ADD
RGBA (or 4)	$R = min(1, R_f + R_t)$ $G = min(1, G_f + G_t)$ $B = min(1, B_f + B_t)$ $A = A_f A_t$

Texture Combine Environment Mode

- Based upon GL_ARB_texture_env_combine
- New texture environment function (COMBINE_ARB) allows programmable texture combiner operations
- Operations are decoupled between a color (RGB) and an alpha (A) portions
- Optional Scale factor performed on the result of the operation, I.e. 1,2,4.
- Optional Input mapping for each portion:
  - RGB: SRC_COLOR, ONE_MINUS_SRC_COLOR, SRC_ALPHA, ONE_MINUS_SRC_ALPHA
  - Alpha: SRC_ALPHA, ONE_MINUS_SRC_ALPHA

Texture Combine Environment Mode

- **combiner operations**

REPLACE	MODULATE	ADD
Arg0	*Arg0 * Arg1*	*Arg0 + Arg1*

Where *Arg0, Arg1 and Arg2 are derived from:*

PRIMARY_COLOR_ARB	primary color of incoming fragment
TEXTURE	texture color of corresponding texture unit
CONSTANT_ARB	texture environment constant color
PREVIOUS_ARB	result of previous texture environment; on texture unit 0, this maps to PRIMARY_COLOR_ARB

Texture Combine Environment Mode

ADD_SIGNED_ARB	SUBTRACT_ARB	INTERPOLATE_ARB
Arg0 + Arg1 - 0.5	*Arg0 - Arg1*	*Arg0 * (Arg2) + Arg1 * (1 - Arg2)*

Where *Arg0, Arg1 and Arg2 are derived from:*

PRIMARY_COLOR_ARB	primary color of incoming fragment
TEXTURE	texture color of corresponding texture unit
CONSTANT_ARB	texture environment constant color
PREVIOUS_ARB	result of previous texture environment; on texture unit 0, this maps to PRIMARY_COLOR_ARB

Texture Dot3 Environment Mode

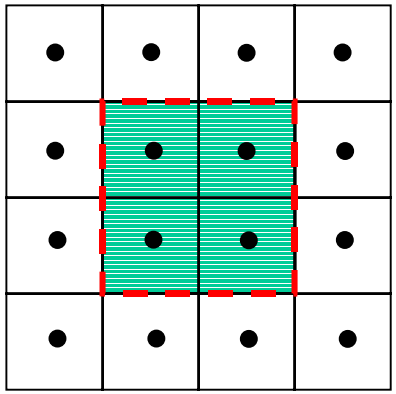
- **Based upon GL_ARB_texture_env_dot3**
- **Adds a dot product operation to GL_ARB_texture_env_combine**
- **The dot product operates on the RGB components of the source arguments and "smears" the results across the RGB channels (DOT3_RGB_ARB) or RGBA (DOT3_RGBA_ARB) of the destination**

Texture Dot3 Environment Mode

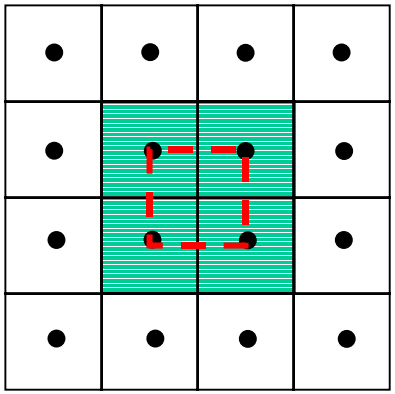
DOT3_RGB_ARB	DOT3_RGBA_ARB
R=G=B= *4*((Arg0_r - 0.5)*(Arg1_r - 0.5) +* *(Arg0_g - 0.5)*(Arg1_g - 0.5) +* *(Arg0_b - 0.5)*(Arg1_b - 0.5))*	*R=G=B=A=* *4*((Arg0_r - 0.5)*(Arg1_r - 0.5) +* *(Arg0_g - 0.5)*(Arg1_g - 0.5) +* *(Arg0_b - 0.5)*(Arg1_b - 0.5))*

Texture Border Clamp

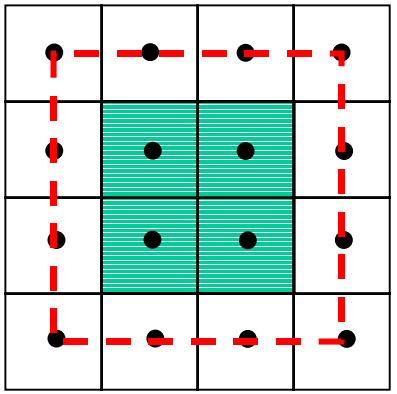
- **Based on GL_ARB_texture_border_clamp**
  - **causes texture coordinates to be clamped to a range 1/2 texel outside [0, 1]; this prevents the "half border, half edge" color artifact**



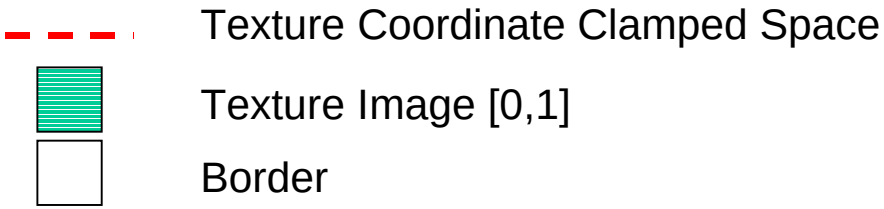
GL_CLAMP



GL_CLAMP_TO_EDGE



CLAMP_TO_BORDER_ARB



Multisample

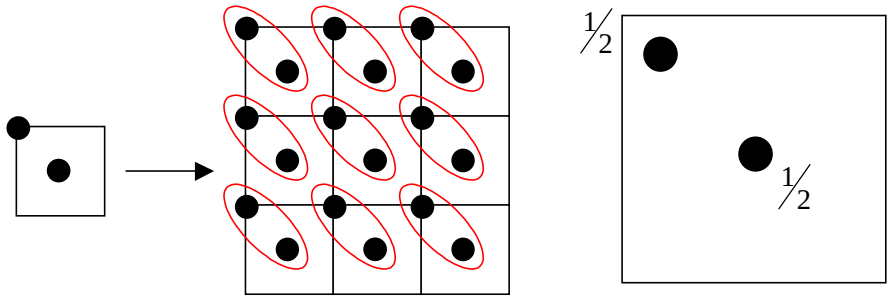
- Based upon WGL_ARB_multisample
- Requires WGL_ARB_extensions_string and WGL_ARB_pixel_format
- Allows the application to get a multisampled frame buffer with a given number of samples per pixels

- Multisample filtering taxonomy:
  - Sample : a subpixel frame buffer sample containing color, depth, and stencil information
  - Tap : source of data for filtering

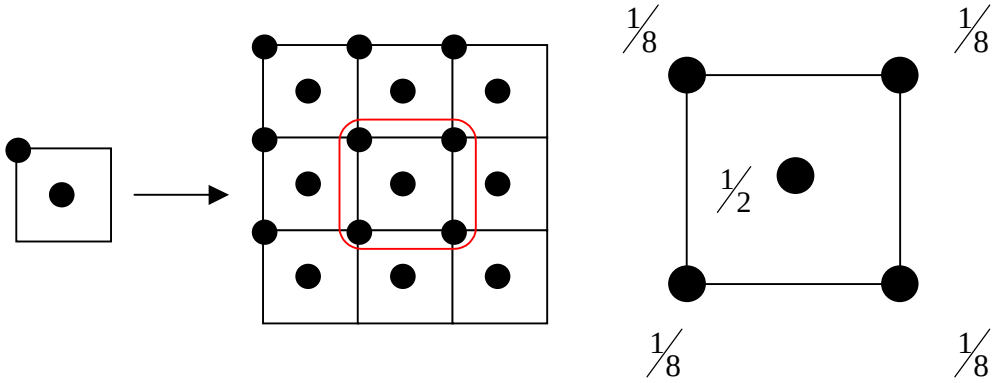
Multisample

- **GeForce3 2 sample filter kernels**

2 Sample
2 Tap
Box
Multi-Sampling



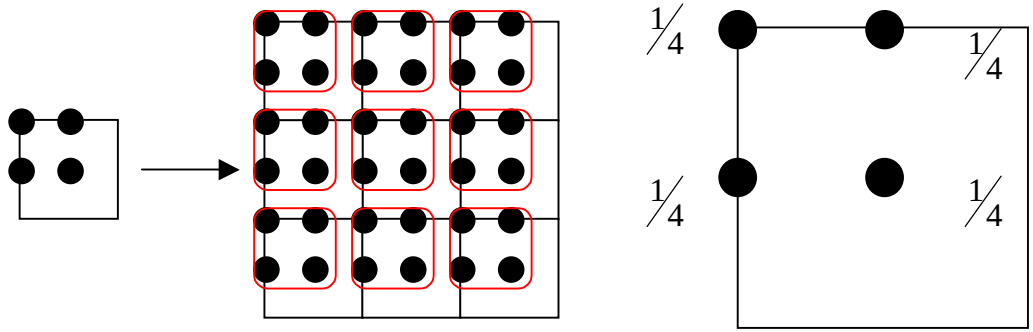
2 Sample
5 Tap
Quincunx
Multi-Sampling



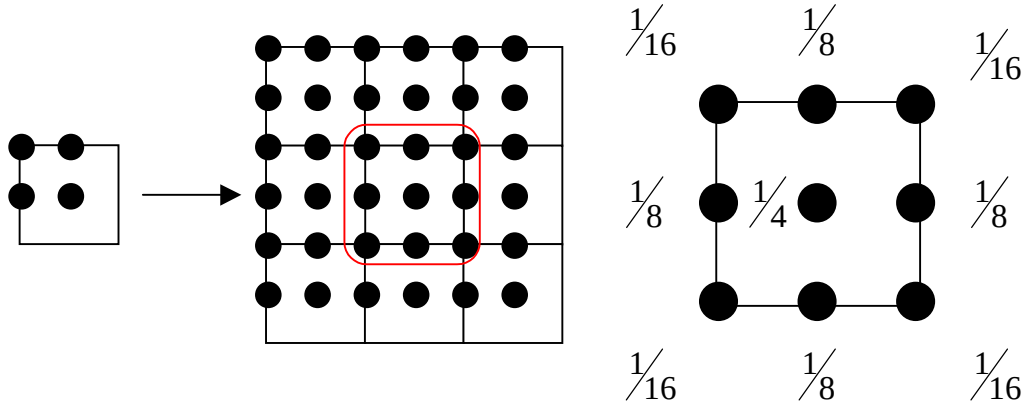
Multisample

- **GeForce3 4 sample filter kernels**

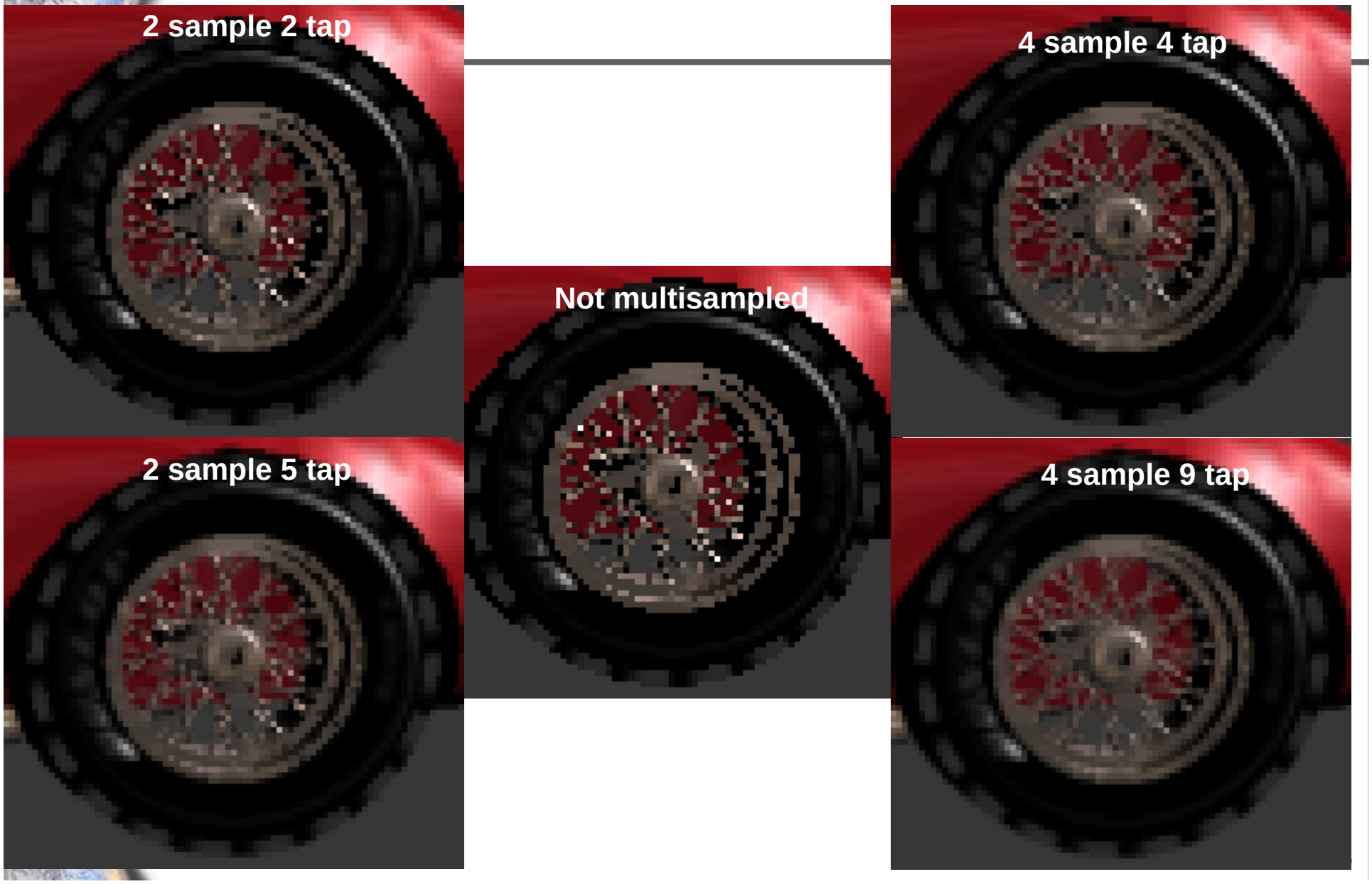
4 Sample
4 Tap
Box
Multi-Sampling



4 Sample
9 Tap
Box
Multi-Sampling



Multisample



Transpose Matrix

- **Based upon GL_ARB_transpose_matrix**
- **Allows to specify row-major order matrices to GL**
  - **Matches the C-language memory layout**

$$\begin{pmatrix} 0 & 4 & 8 & 12 \\ 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \end{pmatrix}^{T} = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{pmatrix}$$

Perspectives…

- Today's Graphics Hardware is
  - programmable:
    - Vertex
    - Fragment
  - precise
    - Vertex Programs – 32 bit IEEE floating point per component
    - Texture Shaders – 32 bit IEEE floating point per component
- Tomorrow's Graphics Hardware will have
  - More programmability…
  - More precision…

Questions?

Send to sdomine@nvidia.com