4th Generation PC-class Programmable GPU Architecture

The Direct3D 10 System

Introduction

- Traditional rendering pipeline
  - Modeling transformation
  - Per-vertex Lighting
  - Viewing transformation
  - Projection transformation
  - Clipping
  - Rasterization
  - Texturing, Per-Fragment Lighting

Introduction

- Hardware accelerated rendering pipeline
  - Modeling transformation
  - Per-vertex Lighting
  - Viewing transformation
  - Projection transformation
  - Clipping
  - **Rasterization**
  - **Texturing, Per-Fragment Lighting**

H/W rasterizer

Introduction

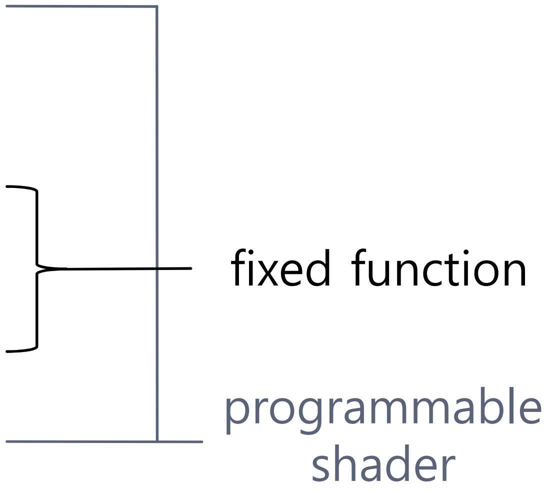
- Hardware accelerated rendering pipeline
  - Modeling transformation
  - **Per-vertex Lighting**
  - **Viewing transformation**
  - **Projection transformation**
  - **Clipping**
  - **Rasterization**
  - **Texturing, Per-Fragment Lighting**

T&L acceleration

H/W rasterizer

Introduction

- Programmable H/W rendering pipeline
  - Modeling transformation
  - **Programmable Vertex Shader**
  - **Clipping**
  - **Rasterization**
  - **Programmable Pixel Shader**

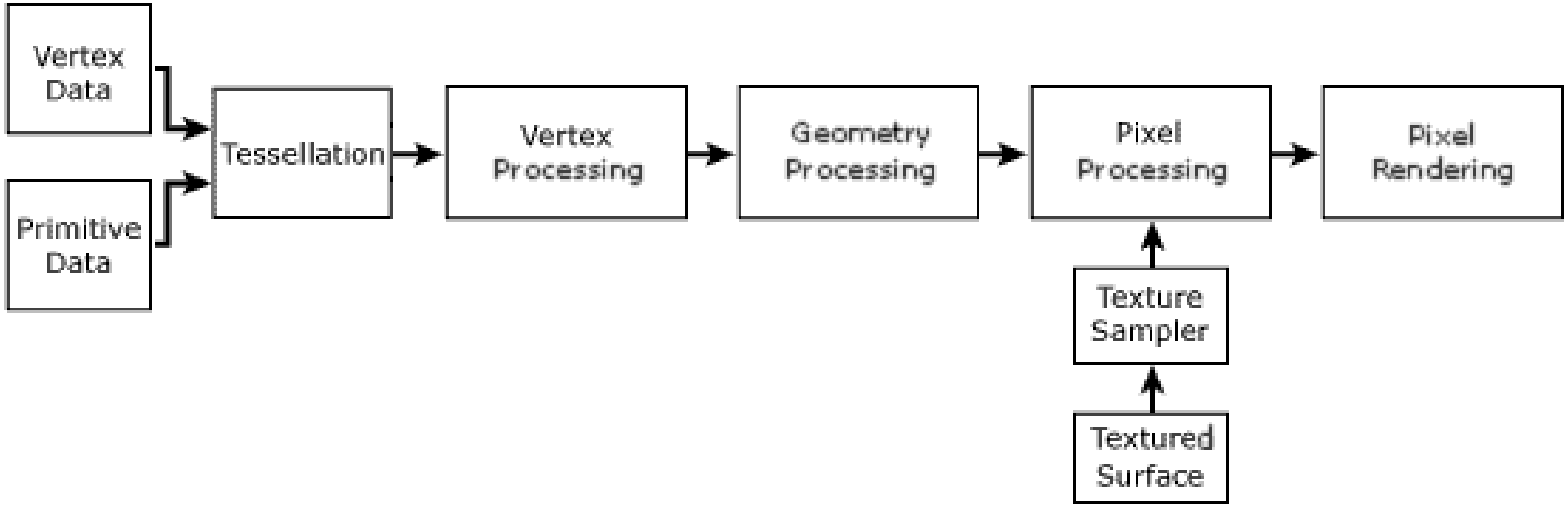


fixed function

programmable shader

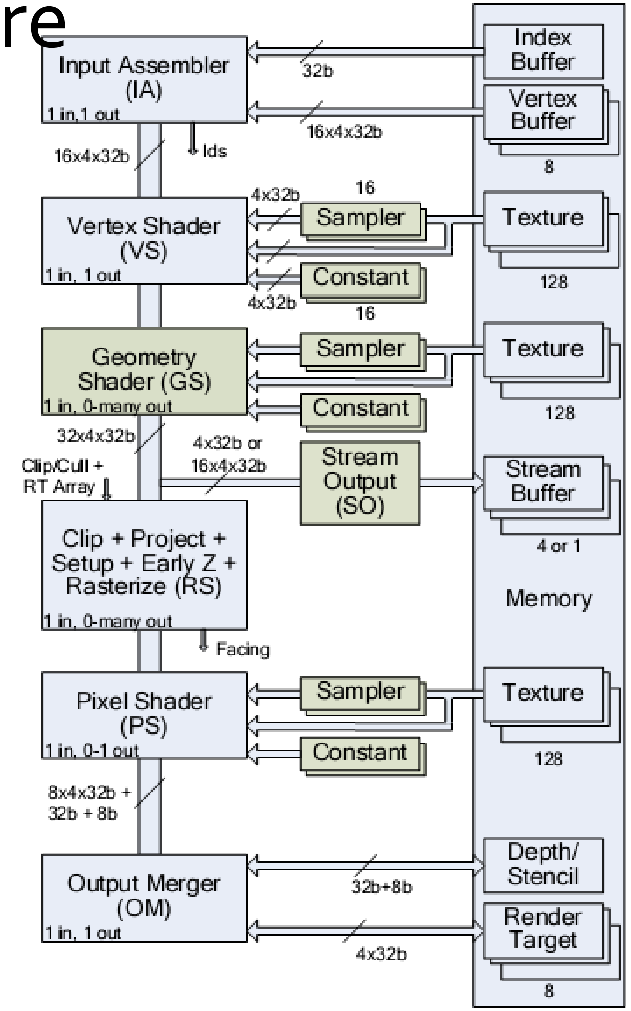
Introduction

- Direct3D 9 graphics pipeline



Direct3D 10 Pipeline

- Retains the traditional structure
- New features
  - Instancing on IA stage
- New stages
  - Geometry Shader (GS)
  - Stream Output (SO)

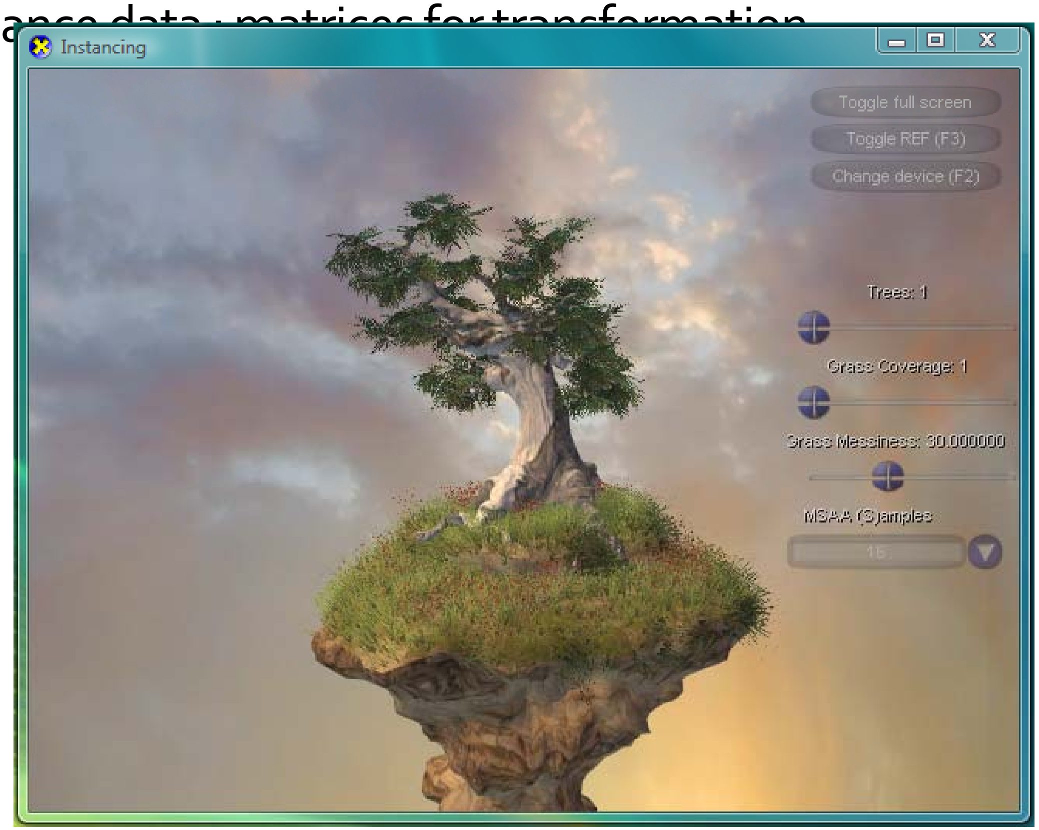


Instancing

- Produces an object multiple times
  - with some variation
  - Separation of per-vertex and per-instance data
- Applications
  - Object with hierarchical self-similarity
  - Mass simple objects / Particle System

Instancing

- Example : Tree Generation
  - Per-vertex data : positions, normal, textures
  - Per-instance data : matrices for transformation



Trees: 1

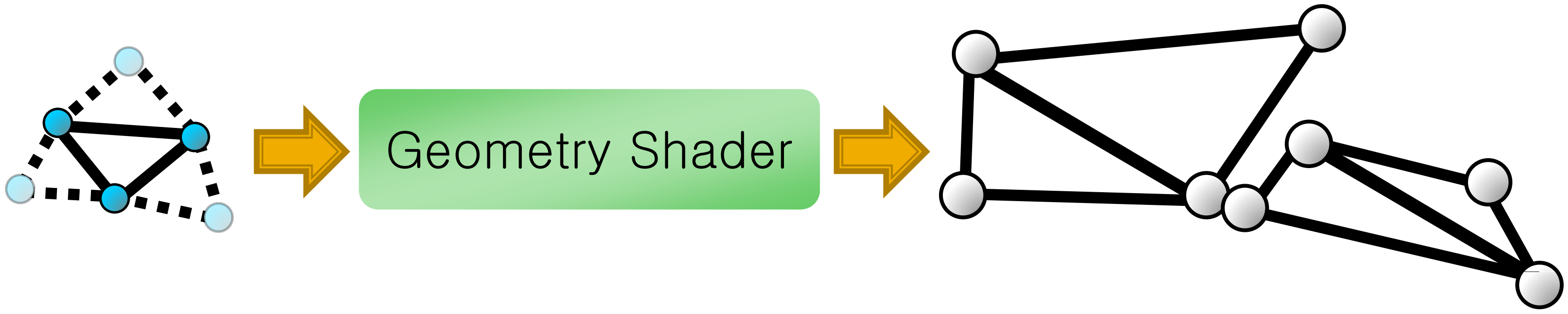
Grass Coverage: 1

Grass Messiness: 30.000000

MSAA (S)amples 16

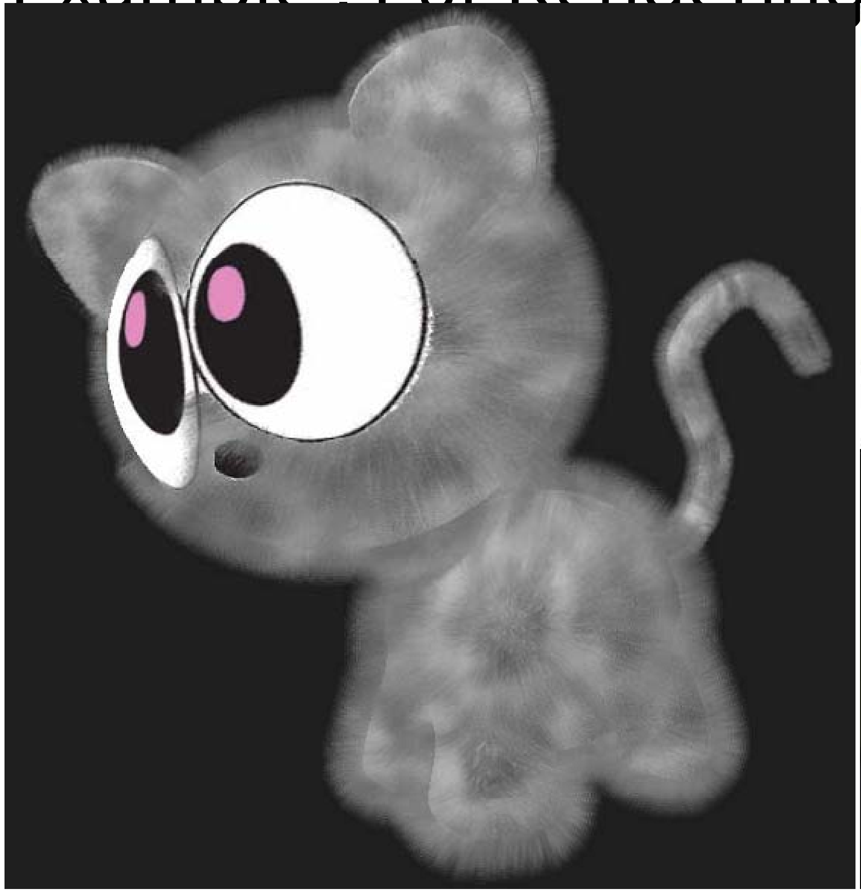
Geometry Shader

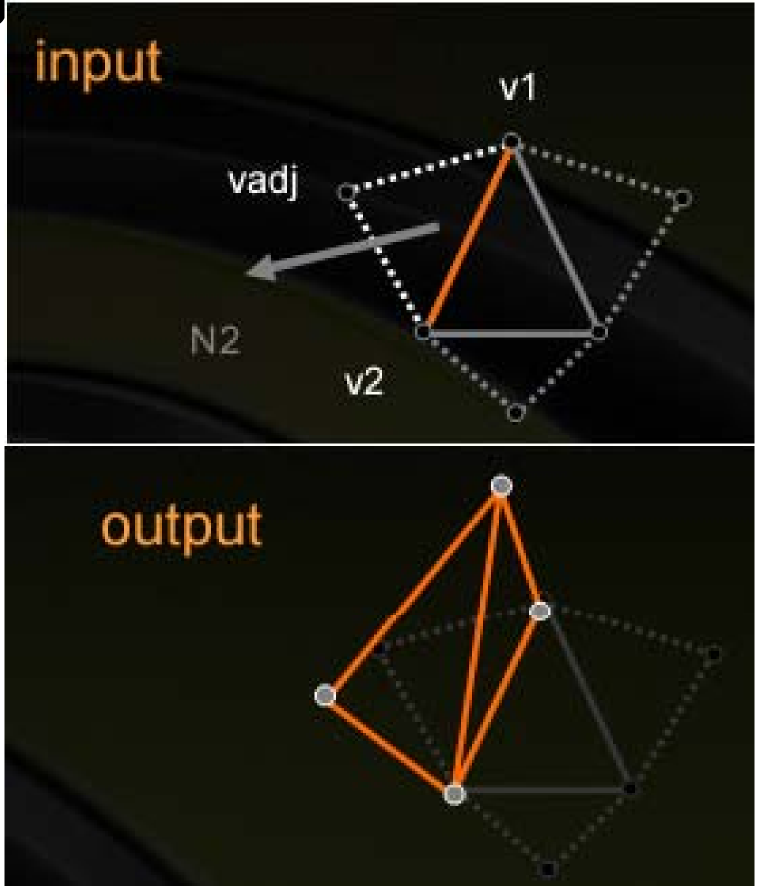
- Primitive-based processing
  - One primitive as input
  - 0~many primitives as output



Geometry Shader

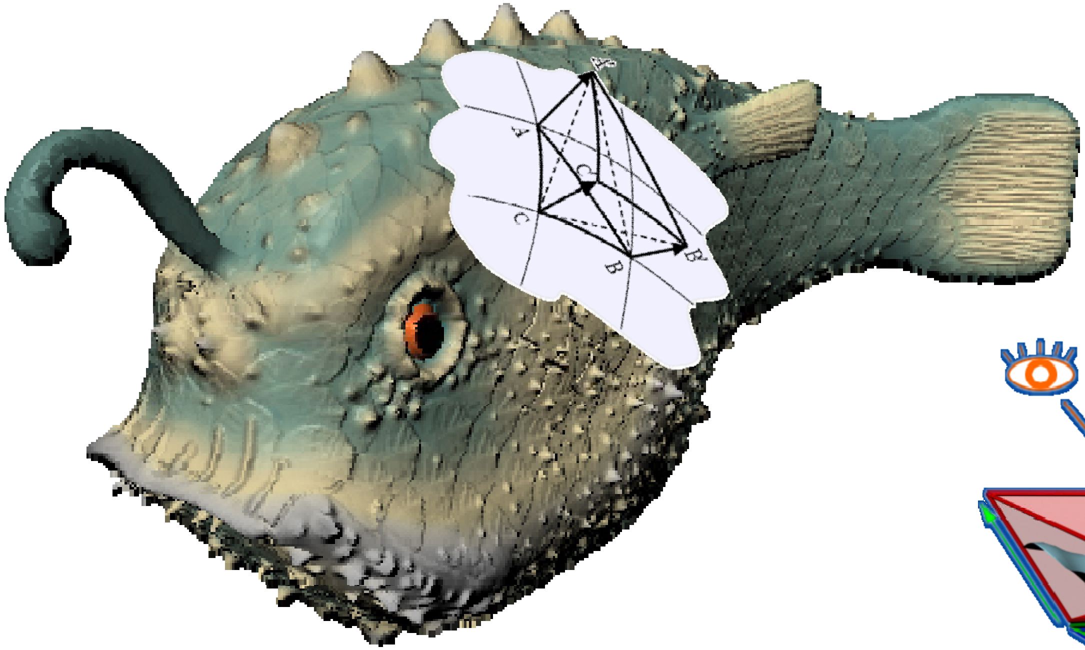
- Example : Fur Rendering

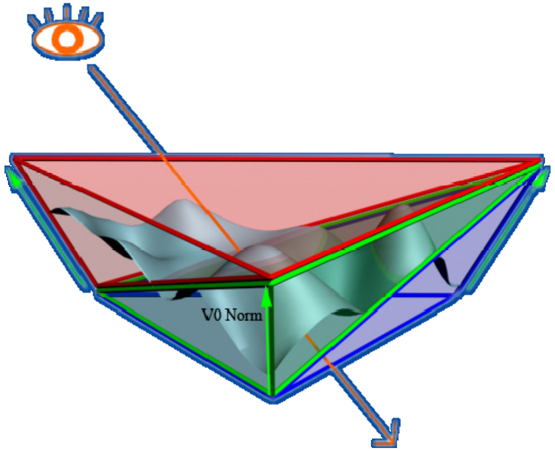




Geometry Shader

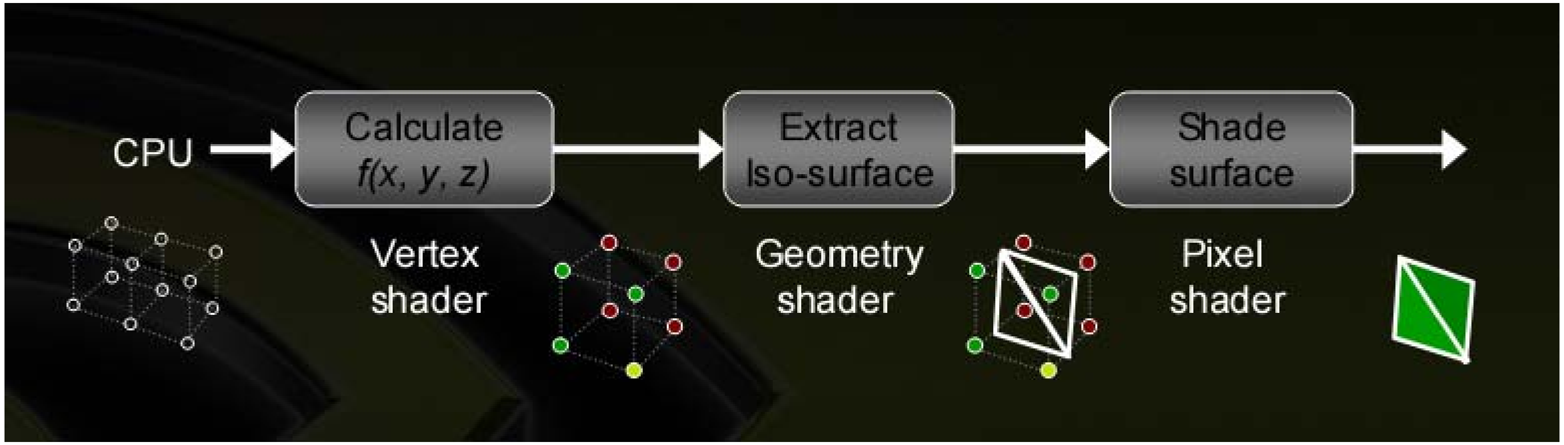
- Example : Displacement Mapping





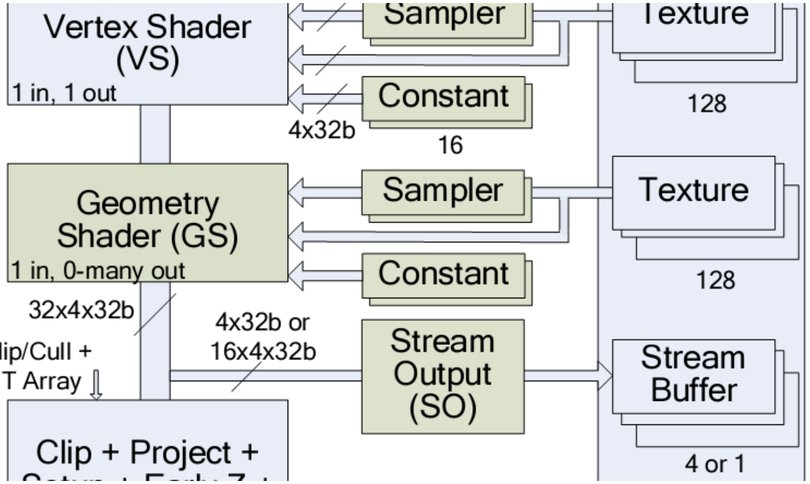
Geometry Shader

- Example : Marching Cubes
  - Grid of vertices
  - GS processes each cube in turn



Stream Output

- Allows storing output from GS or VS
  - Four 1D buffers
- Sequential 1D output
- Enables multi-pass geometry operations
  - Recursive Subdivision



Memory Structure and Data Flow

- Resource
  - Generalized data structure
  - Two type : ‘buffer’ and ‘texture’
  - Arrayable
  - Polymorphism : View
    - Render to a vertex buffer
    - Render to a 2D slice of 3D texture
    - Texel sampling from a vertex buffer
  - Constraints have been relaxed, but still not universal

Storage Formats

name	**widths**	**range**
unormN, snormN	8, 16	[0,1] and [-1,1]
floatN	32, 16, 11, 10	s23e8, s10e5, 6e5, 5e5
uintN, sintN	8, 16, 32	$[0,2^n-1]$ and $[-2^{n-1},2^{n-1}-1]$
RGBE	32	9/9/9e5
SRGB	8	[0,1] non-linear

Resource Mapping and Access

- CPU-GPU access to resources
  - Most complex issues on GPU design
  - Bandwidth problem
    - GPU - Video memory : more than 50GB/s
    - GPU - System memory : 2.8GB/s (PCI-express)
  - Read/Write waiting lock
    - Dramatic effects on performance

Resource Mapping and Access

- Lock/unlock → Map/unmap
- Update functions
  - Copy from a resource to another resource
- Four resource usages

	GPU access	CPU access
Default	Read/Write	Denied
Immutable	Read-only	Denied
Dynamic	Read/Write	Write-only
Staging	Denied	Read/Write

What to do?

- Volume Rendering
  - Encoding / Compression
  - HDR, NPR rendering
  - Polygon & Volume rendering
- Image Processing
  - SRG
  - Level Set Method
  - Other issues?