Computer Graphics Charles University

Introduction to OpenGL

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

http://cgg.mff.cuni.cz/~pepca/

Advances in Hardware

- **3D acceleration** is a common feature in consumer devices
- Focus on **games**, **multimedia**
- **Appearance** – quality of results
  - Sophisticated texturing functionality
  - Combinations of many textures and processing
- High **performance**
  - The best semi-conductor technology is used for GPUs (NVIDIA Kepler: 28 nm), **massive parallelism**
  - Very fast **memory** (multipath access, GDDR5, ..)
  - High performance bus between GPU and CPU (PCI-E)

Advances in Software

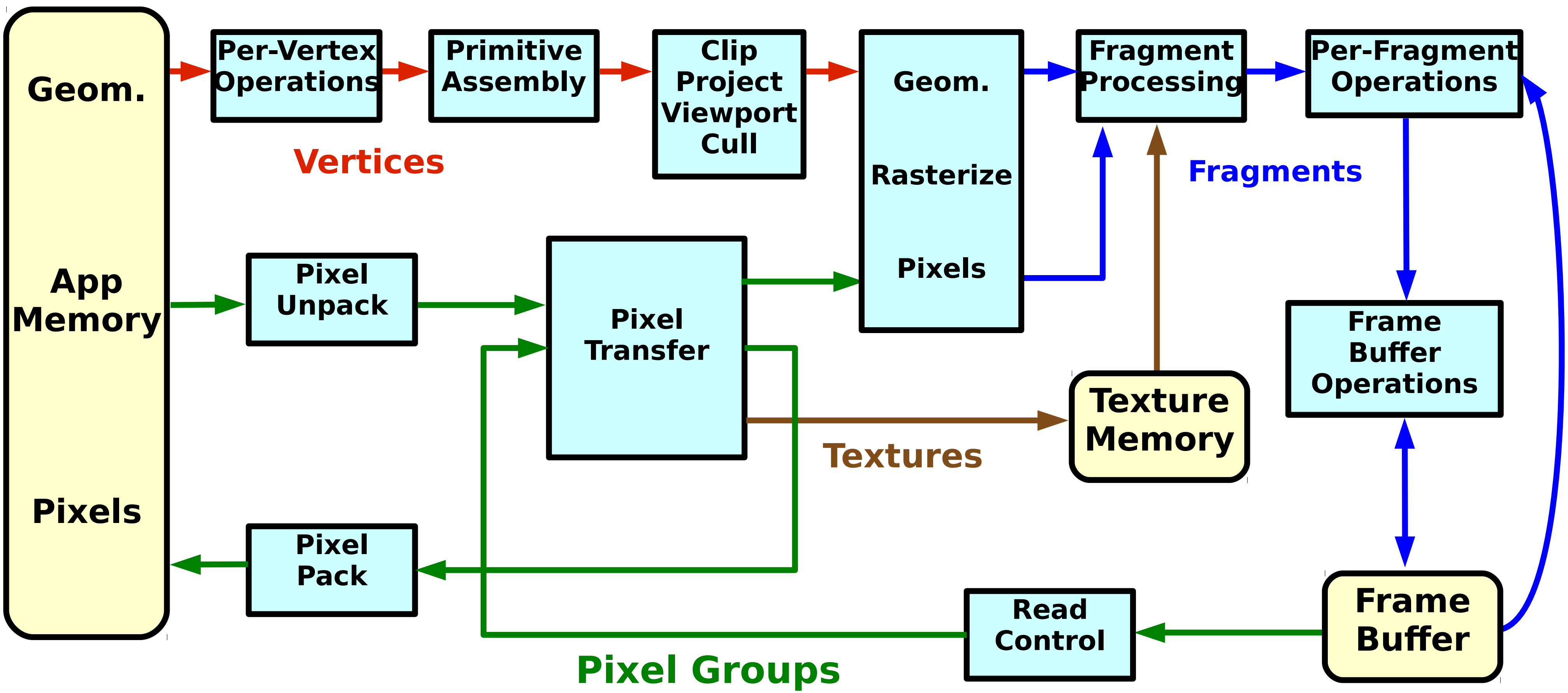
- Two main **libraries/APIs** for 3D graphics
  - **OpenGL** (SGI, open standard) and **Direct3D** (Microsoft)
  - Approach is similar, API is hardware-determined
- **Efficient transmission of data** to the GPU
  - Maximal use of shared data fields
- **Programmable shaders!**
  - Revolution in graphics programming
  - „***vertex-shader***": mesh vertex processing
  - „***geometry/tesselation-sh.***": generating geometry
  - „***fragment-shader***" („*pixel-shader*"): executed for each pixel that is displayed

Development Tools

- Can be used by programmers and artists
- **Higher languages** for GPU programming
  - Cg (NVIDIA), HLSL (DirectX), GLSL (OpenGL)
  - Cg and HLSL are very similar
- **Composition of graphical effects**
  - Compact description of the entire effect (GPU programs, references to data) in one file
  - DirectX **.FX** format, NVIDIA **CgFX** format
  - Tools: Effect Browser (Microsoft), **FX Composer** (NVIDIA), **RenderMonkey** (ATI)

OpenGL Schematics (FFP)

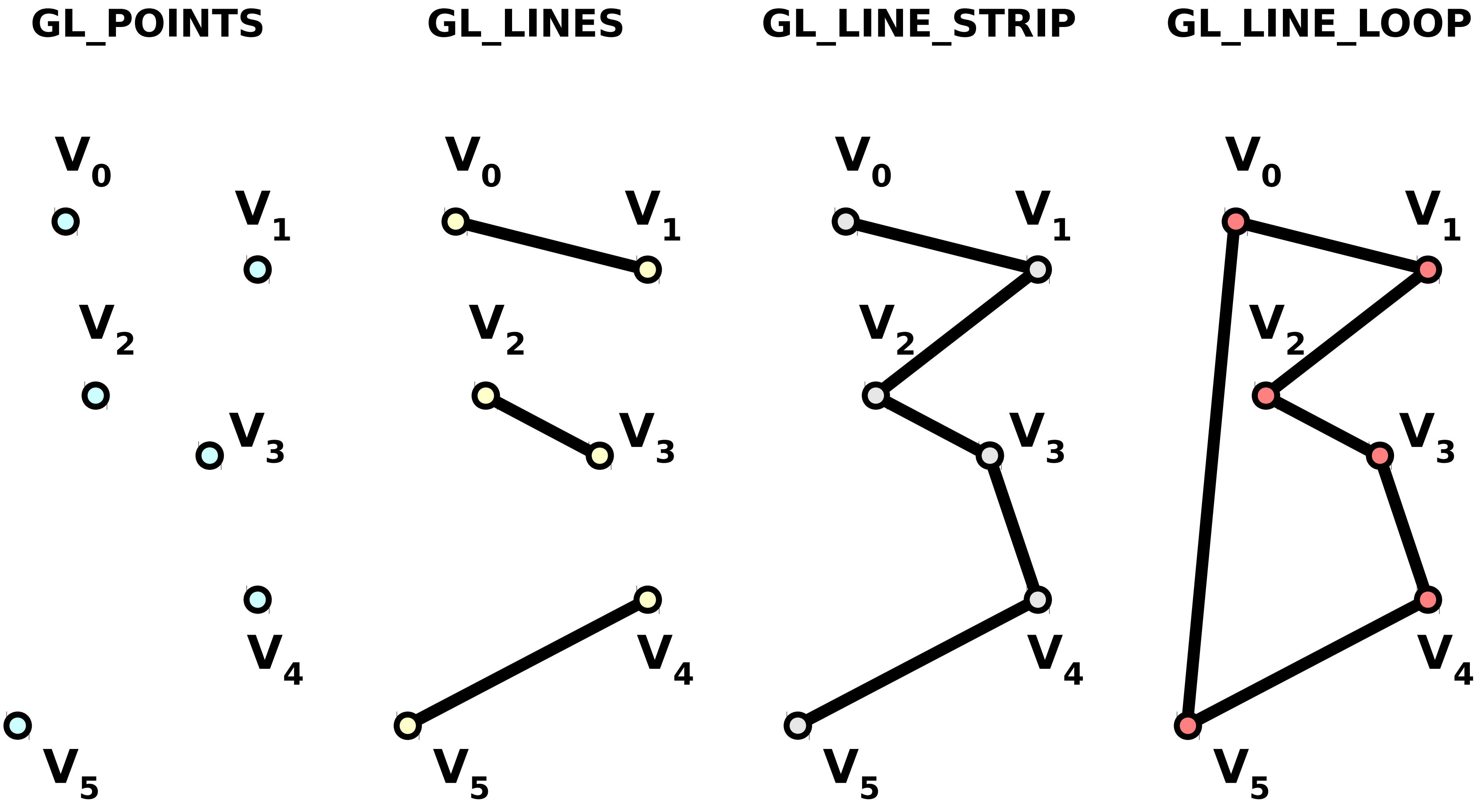
- OpenGL Fixed Functionality Pipeline:



OpenGL: Geometric Primitives

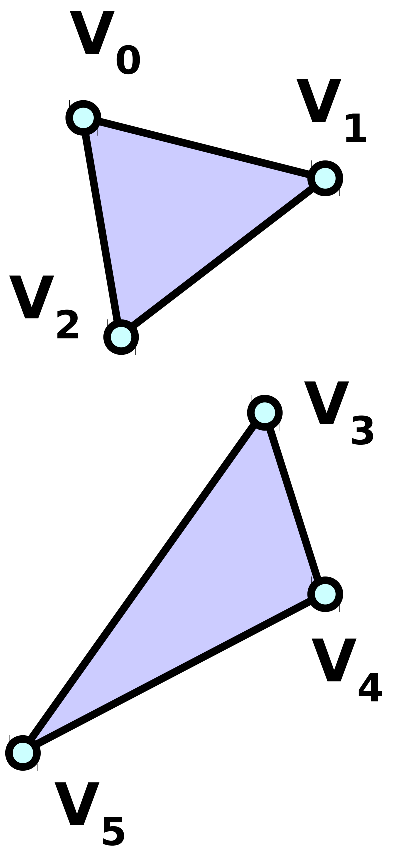
- Types of geometric primitives:
  - **point**, **line**, polyline, loop
  - polygon, **triangle**, triangle strip, triangle fan, quad, quad strip
- Processing of **individual vertices**
  - glVertex, glColor, glNormal, glTexCoord, ...
  - <u>inefficient</u> (many calls to gl* functions)
- **Vertex Arrays**
  - glDrawArrays, glMultiDrawArrays, <u>glDrawElements</u>, ...
  - glColorPointer, glVertexPointer, ... or **interlaced**

Geometric Primitives I

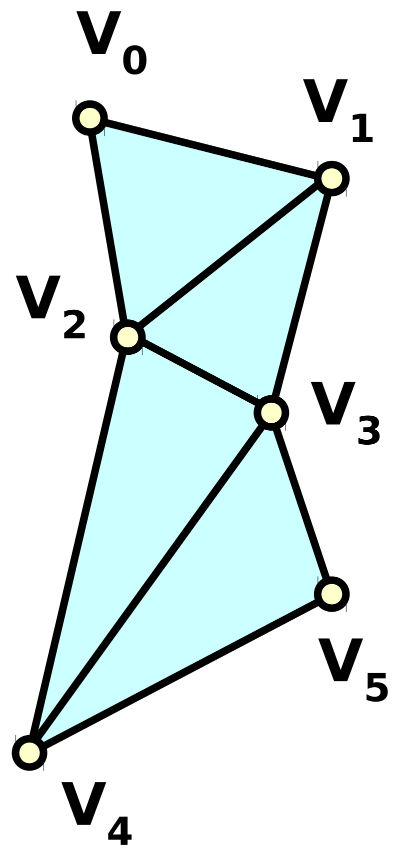


Geometric Primitives II

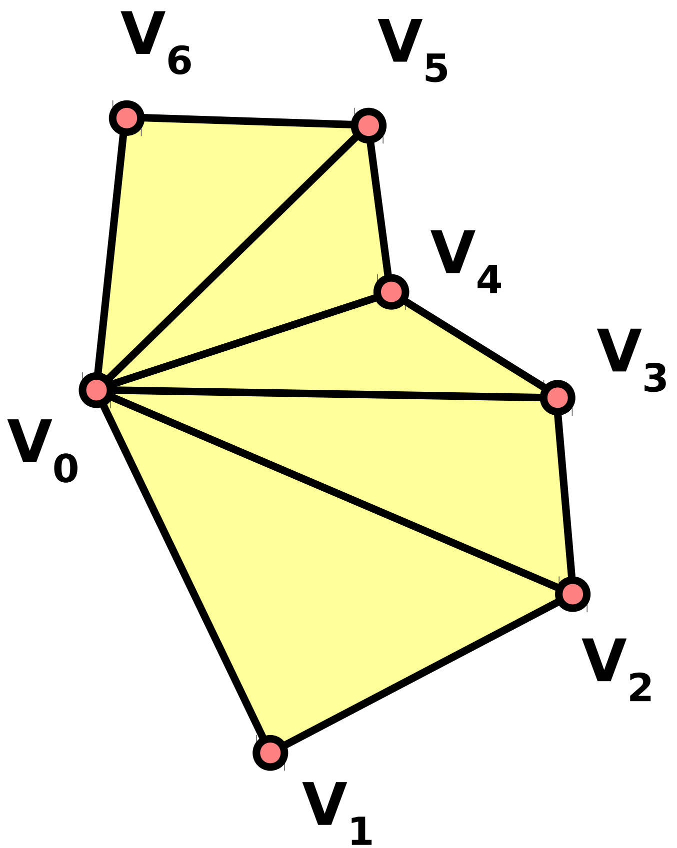
GL_TRIANGLES



GL_TRIANGLE_STRIP

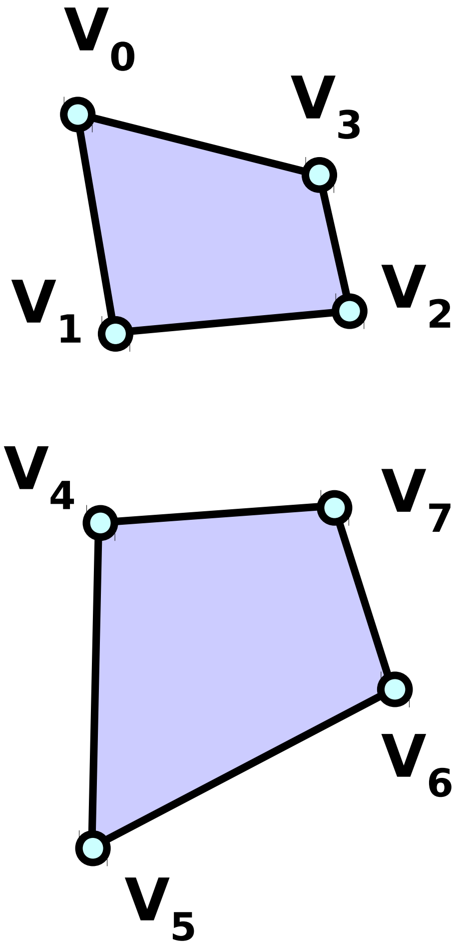


GL_TRIANGLE_FAN

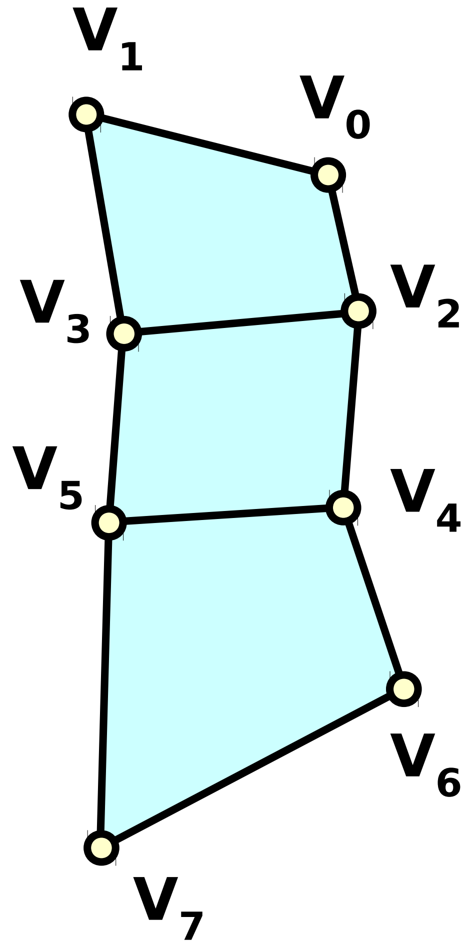


Geometric Primitives III

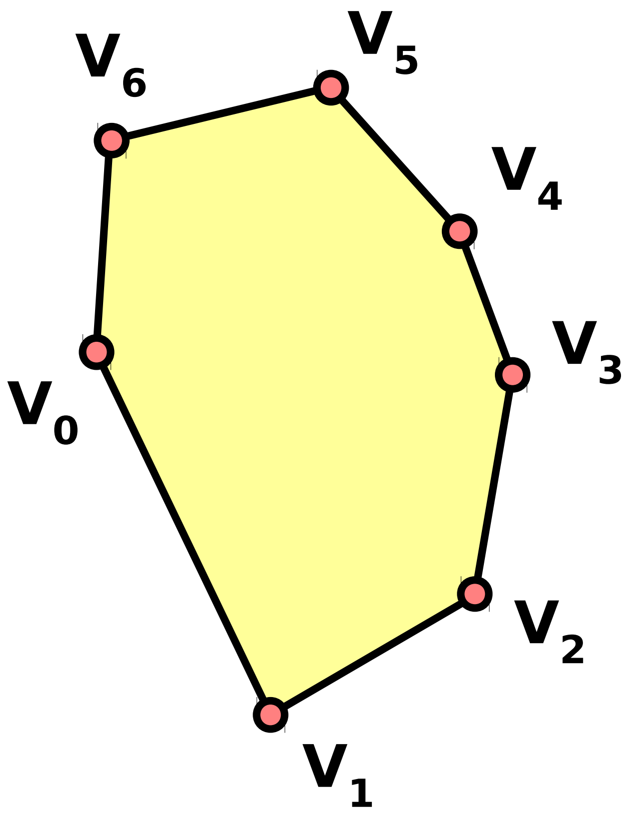
GL_QUADS



GL_QUAD_STRIP



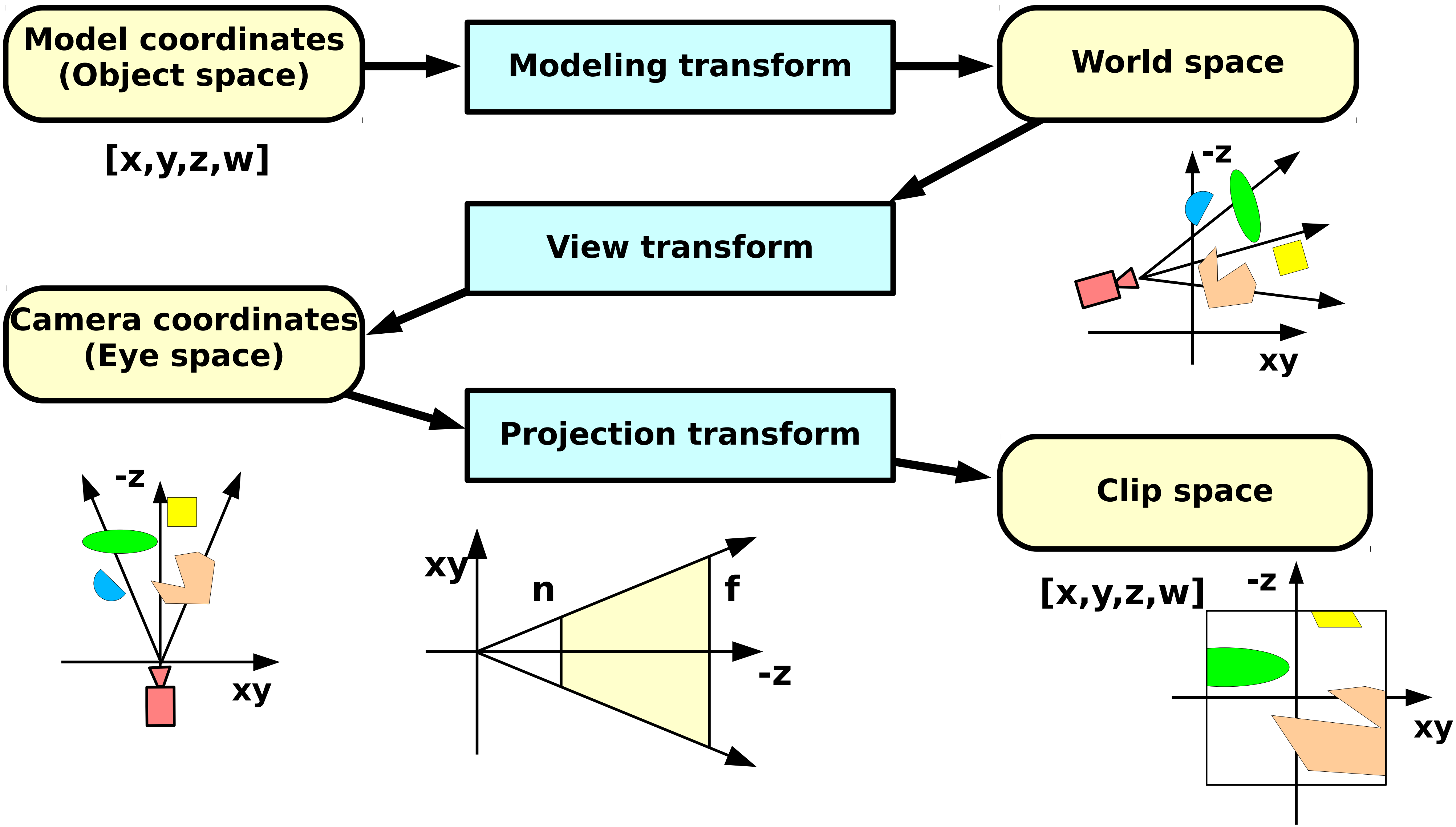
GL_POLYGON



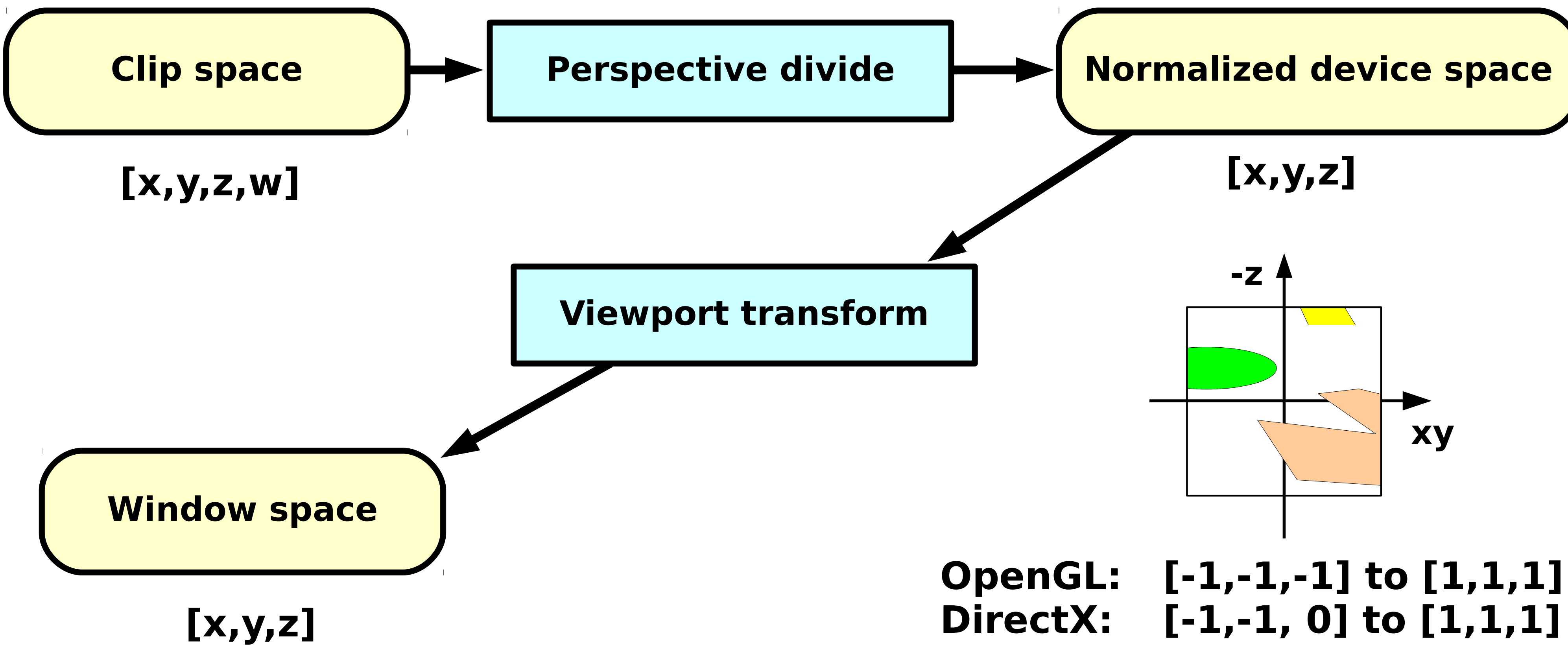
OpenGL „macros“ (Display Lists)

- DISPLAY_LIST_MODE instead of IMMEDIATE_MODE
- **Storage** of a sequence of GL commands in memory:
  - glNewList, glEndList
  - The list can be pushed to the server (graphical HW)
  - Basic idea: „list = macro”
- **Execution** of entire lists
  - glCallList, glCallLists
  - Potentially very <u>efficient</u> (the commands can be optimised by the graphics server)

Coordinate Systems



Coordinate Systems II



OpenGL: [-1,-1,-1] to [1,1,1]
DirectX: [-1,-1, 0] to [1,1,1]

[x,y] actual fragment size on screen
z depth compatible with the z buffer

Coordinate Systems III

- **Model coordinates**
  - Database of objects that comprise the scene
  - Source: 3D modeling applications (3DS, Maya, ..)
- **World coordinates**
  - Absolute coordinates of the 3D world
  - The relative coordinates of <u>object instances</u> are given there
- **Camera coordinates**
  - 3D world → relative camera coordinates
  - Projection center: **origin**, view direction: **-z** (or **z**)

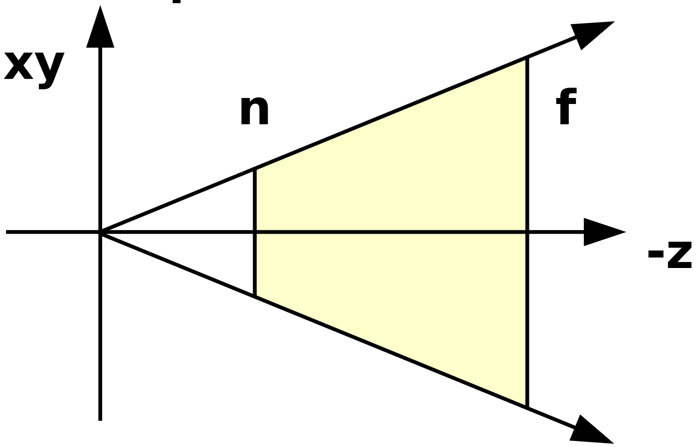
Coordinate transforms

- **Model → camera transforms**
  - Combined transformation „modelview”
  - World coordinates are not directly used
- **Projection transformation**
  - Defining the view **frustum** [**l**, **r**, **b**, **t**, **n**, **f**]
  - Near and far clipping plane: **n**, **f**
  - Result in homogeneous coordinates (before clipping)
- **Clip space**
  - **Output coordinates** of vertex shaders!

Projective Transformation

- The remote point **f** can be moved to <u>infinity</u>

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ -\frac{r+l}{r-l} & -\frac{t+b}{t-b} & \frac{f+n}{f-n} & 1 \\ 0 & 0 & -\frac{2fn}{f-n} & 0 \end{bmatrix} \qquad \begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ -\frac{r+l}{r-l} & -\frac{t+b}{t-b} & 1 & 1 \\ 0 & 0 & -2n & 0 \end{bmatrix}$$

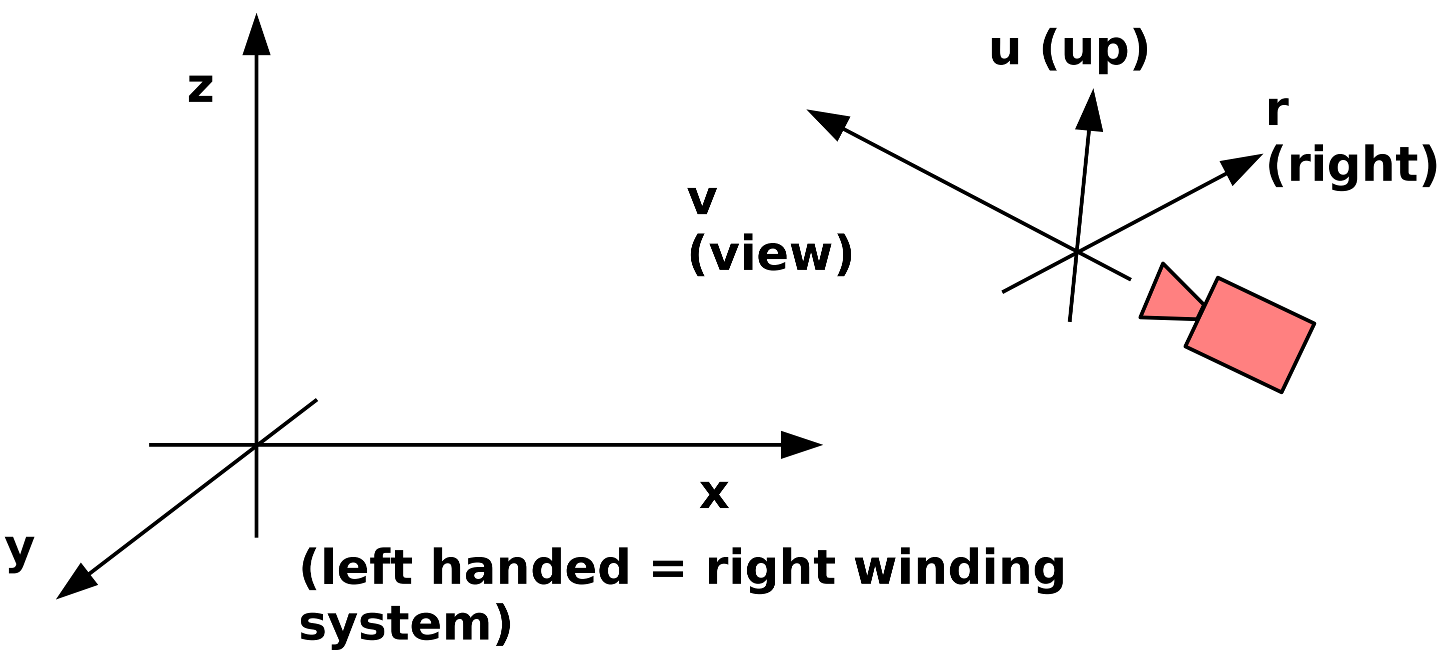


Coordinate Transformations

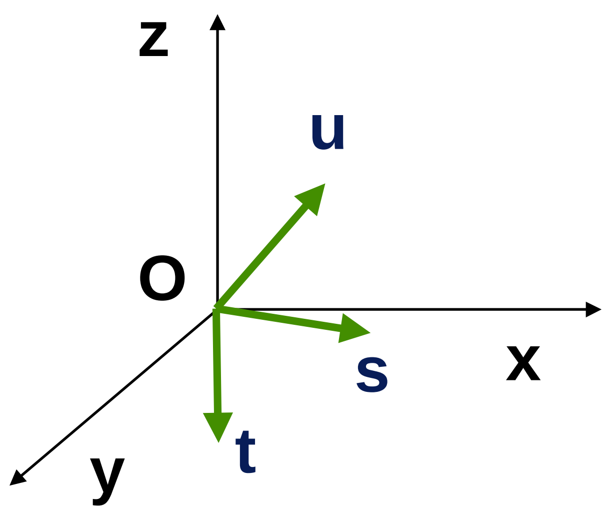
- **Perspective division/clip**
  - Transforms homogeneous coordinates to cartesian
- **Normalised Device Coordinates** („**NDC**”)
  - Unit sized cube
  - OpenGL: [-1,-1,-1] to [1,1,1]
  - DirectX: [-1,-1,0] to [1,1,1]
- **Window coordinates** („window space”)
  - Result of screen and depth transformation
  - Used during **rasterisation** and when working with **fragments**

Rigid Body Transformations

- Retain **shape**, only change **orientation**
  - Only consists of translation and rotation
  - Often used for **coordinate system transforms** (e.g. world coordinates, and viewer coordinates)



Transformation Between Systems



A **coordinate system** has its origin in **O** and is given by three basis vectors

[s, t, u]

$$M_{stu \to xyz} = \begin{bmatrix} s_x & s_y & s_z \\ t_x & t_y & t_z \\ u_x & u_y & u_z \end{bmatrix}$$

$$[1, 0, 0] \cdot M_{stu \to xyz} = s$$

$$[0, 1, 0] \cdot M_{stu \to xyz} = t$$

$$[0, 0, 1] \cdot M_{stu \to xyz} = u$$

$$M_{xyz \to stu} = M^T_{stu \to xyz}$$

Geometric Data on the Server

- **VBO**, **VAO**, starting with **OpenGL 1.5**
  - For .NET practically a requirement (client memory is not fixed)
- The buffers **on the graphics server** contain geometric data
  - Buffer creation: glBindBuffer
  - **Data entry:** glBufferData, glBufferSubData
  - **Mapping** to application memory: glMapBuffer, glUnmap..
- Work with **client memory** <u>or</u> with **buffers**
  - glColorPointer, glNormalPointer, glVertexPointer, ...

Vertex Processing

- **Vertex transformations** via model and perspective matrices
  - glMatrixMode
  - glLoadIdentity, glLoadMatrix, glMultMatrix
  - glRotate, glScale, glTranslate, ...
- **Lighting attributes**
  - glLight, glLightModel, glMaterial

Assembly and Processing

- **Assembly**
  - Determination which vertices a primitive needs
  - Assembly of data packages and uploading
- **Processing** of primitives
  - Clipping
  - Projection to view frustum – removal of „w“
  - Projection and clipping to 2D viewport
  - Back face culling
    - Single vs. double sided primitives

Rasterisation, Fragments

- **Rasterisation** = display of vector primitives
  - Decomposition of objects into **fragments**
  - Objects: points, lines triangles, bitmaps

- **Fragments**
  - **Raster element**, that potentially contributes to the colour of a pixel
  - Size: pixel size, or smaller (anti-aliasing)
  - Data packing in the raster unit of the GPU:
    - In/Output: **x**, **y**, **z** (depth can be changed!)
    - Texture coordinates $\mathbf{t_0}$ to $\mathbf{t_n}$
    - Specular and diffuse colour, fog, user data, ...
    - Output colour in **RGB** and $\boldsymbol{\alpha}$ (frame-buffer op.)

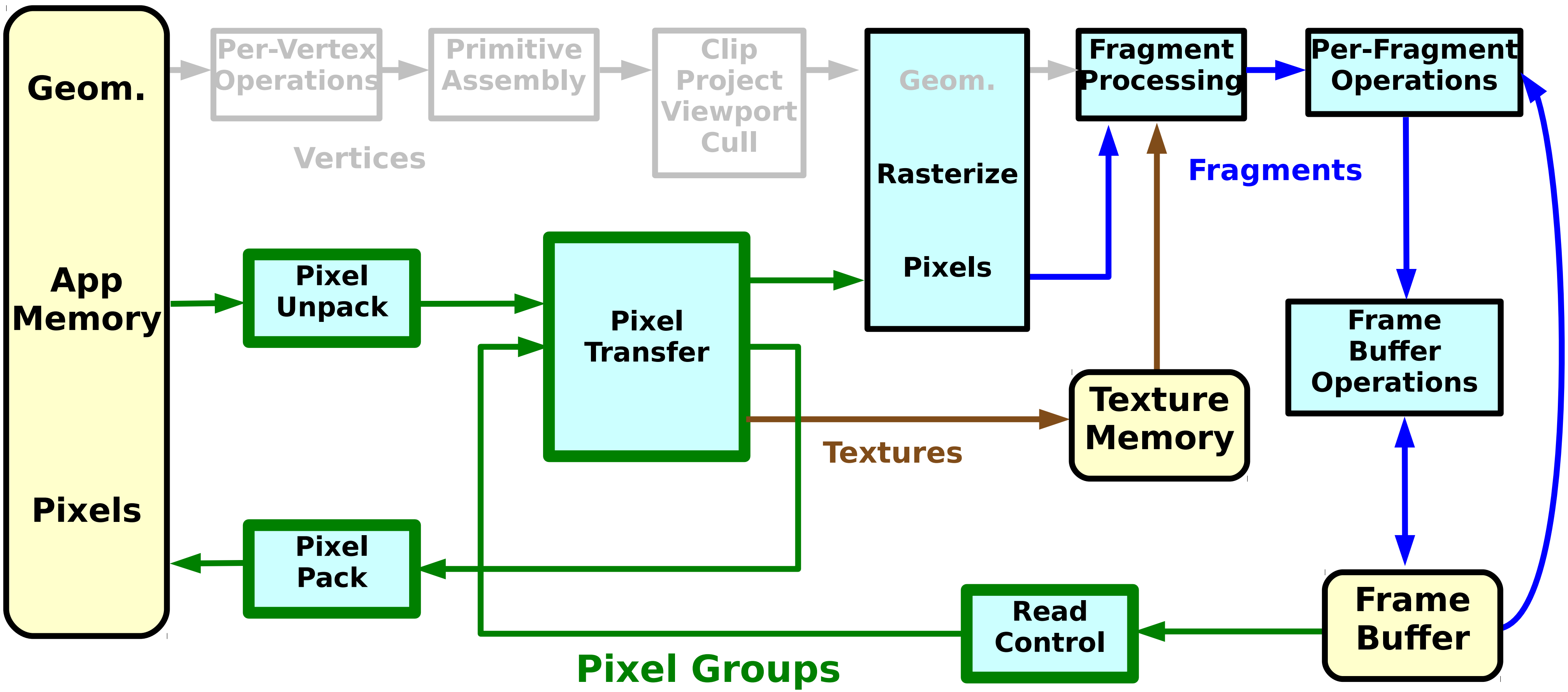
Fragment Interpolation

- Fragment attributes are automatically **interpolated between vertex values for**:
  - depth (**z** or **w**)
  - Texture coordinates
  - Colour (specular and diffuse)
  - User data, ...
- Fast **HW interpolators**
- **Perspectivically correct** interpolation
  - Only [**x**, **y**] change linearly
  - Other variables require one division for each fragment

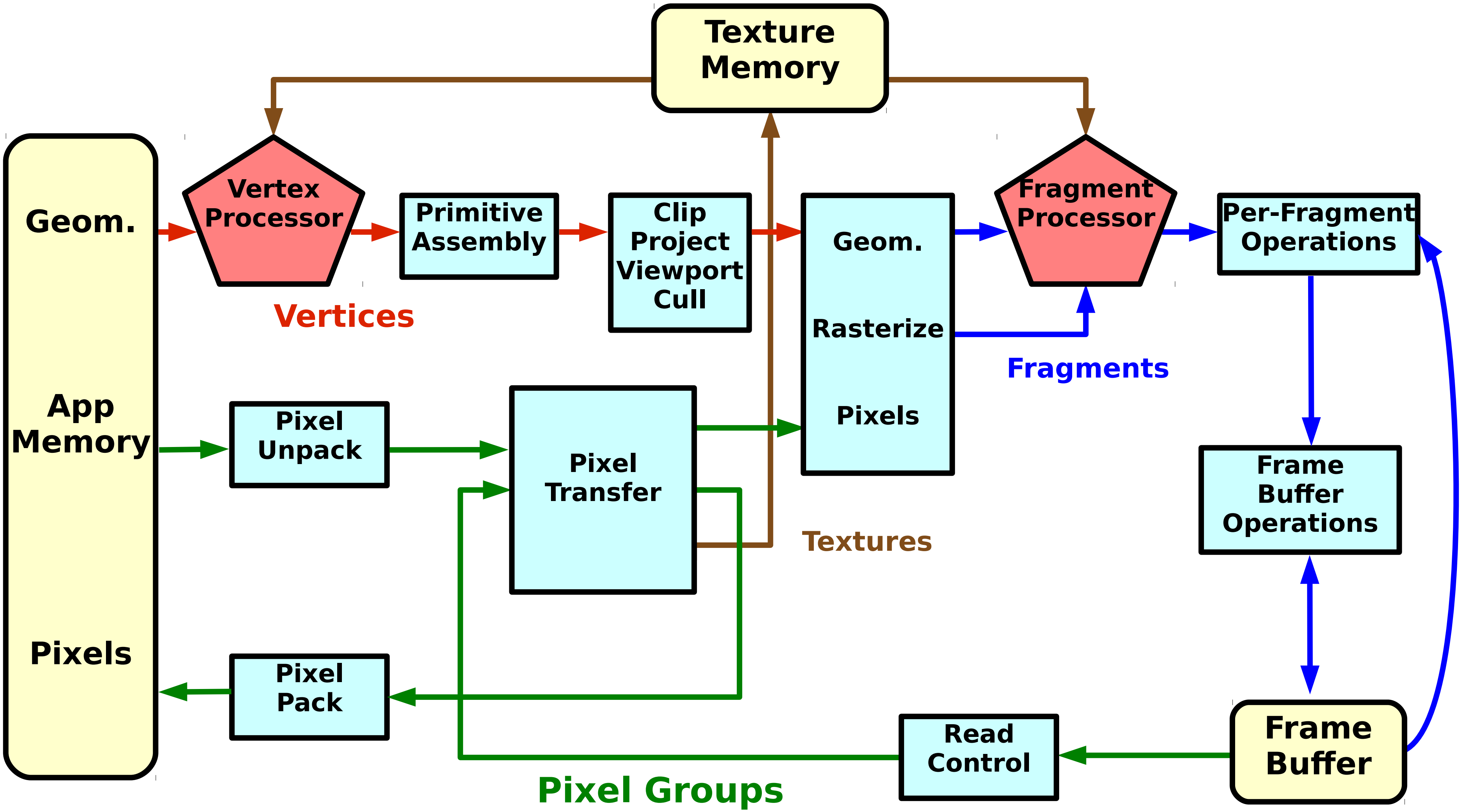
Fragment Processing

- **Texture operations**
  - Highly **optimised** operations
  - Colour selection from texture memory
  - Texel interpolation:
    - mip-mapping, anisotropic filtering, ...
  - Texture combination (many operations possible)
  - Specialised effects (bump-mapping, environment mapping)
- Calculation of **fog**
  - Based on depth **z**
- **Combination** of several colours (diff., spec.)

Raster Images in OpenGL



OpenGL Programmable Pipeline



Vertex Processor

- Replaces the **vertex processing module**:
  - Transformation of vertices
  - Transformation/normalisation of normal vectors
  - Calculation of texture coordinates
  - Calculation of lighting vectors
  - Definition of material constants for vertices
- **What cannot be modified**:
  - **The number of vertices!** (no delete/add)
  - Type & topology of geometric primitives
    - Partial solution: degenerate primitives

Geometric Processor

- **„Tesselation shaders"**
  - New in OpenGL 4.0
  - HW support for subdivision surfaces (spline sheets, ...)
  - Two shaders: „tesselation control" and „tesselation evaluation"
  - The first defines topology, the second generates geometry (coefficients)

- **„Geometry shader"**
  - From OpenGL 3.0 onwards
  - Just before the rasterisation unit
  - Possibility to generate/delete primitives and vertices
  - More powerful than TS, but slower (not easy to incude in schematics)

Fragment Processor

- Replaces the **module for fragment processing**:
  - Arithmetic operations with interpolation
  - Reading of data and textures
  - Texture generation
  - Fog calculations
  - Computation of the final colour of the fragment colour
  - Possibility to modify fragment depth
- **What cannot be modified**:
  - **Number of fragments!** (no addition or deletion)
  - **Fragment position** on screen [x,y]

GPU Programming

- „**Vertex shader**"
  - Code for the vertex processor
- „**Fragment shader**"
  - Code for the fragment processors
- Both can be programmed by the user!
  - **HW independent** programming languages
  - **GPU micro-code** compiled at run-time (hard to work with: many versions, specific to individual cards)
  - Low-level instructions (similar to assembler)
  - or **higher languages** Cg, HLSL, GLSL

Literature

- Tomas Akenine-Möller, Eric Haines: ***Real-time rendering, 2nd edition***, A K Peters, 2002, ISBN: 1568811829
- OpenGL ARB: ***OpenGL Programming Guide, 4th edition***, Addison-Wesley, 2004, ISBN: 0321173481
- Randima Fernando, Mark J. Kilgard: ***The Cg Tutorial***, Addison-Wesley, 2003, ISBN: 0321194969
- Jack Hoxley: ***An Overview of Microsoft's Direct3D 10 API***, 13.12.2005, www.gamedev.net