Universität Bielefeld

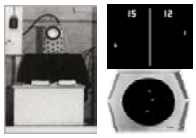
Realtime 3D Computer Graphics Virtual Reality

Graphics

Computer graphics

- 3D-Computer graphics (3D-CG) currently used for
  - Simulators, VR, Games (real-time)
  - Design (CAD)
  - Entertainment (Movies), Art
  - Education

tennis for two
pong
spacewar
(from l. to r.)



Real-time 3D-CG

Goals

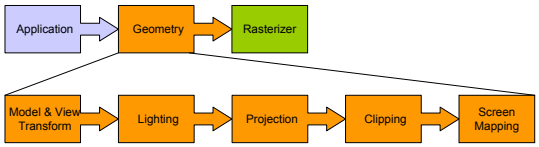
- Generating 3D worlds …
  … that visualize application domain specific data in >3 dimensions (maybe using 3D metaphors).
- Mapping 3D world to 2D pictures …
  … to simulate user-eye-specific views in interactive rates (stereoscopy, immersion).
- Implement user interaction with the virtual world.
- Operate in real-time, minimize lag, maximize simulation rate.

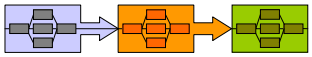
Methods

- Computer graphics methods differ in
  - primitives used and processed (e.g., vertex-, volume-, pixel-based, etc.).
  - lighting model and calculation used (local or global models).
  - direction of "lighting application", shading (scanline shader, raytracer, etc.).
- Trigonometry and linear algebra.

➢ By 2005, the majority of VR-systems produces 2D images based on
  ➢ surface (vertex) models and (image) textures using
  ➢ a local Phong lighting model
  ➢ processed by a **pipeline** architecture
  (changing significantly just now from fixed to programmable pipeline).

The 3D-pipeline

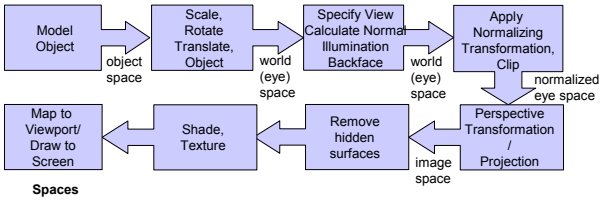




parallelization on different stages

- application stage
- geometry stage
  - model and view transform
  - lighting (shading in rasterizer)
  - projection
  - clipping
  - screen mapping
- rasterizer stage
  - visibility (z-buffer)
  - shading
  - texturing

The 3D-pipeline



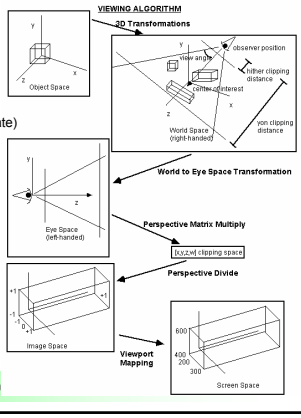
Spaces

1. **Object Space:** Local objects' frames of reference (aka Modeling Space).
2. **World Space:** Where the scene and viewing specification is made.
3. **Eyespace (Normalized Viewing Space):** Eye point (COP) is at the origin looking down the Z axis.
4. **3D Image Space:** A 3D perspected space. Dimensions: -1:1 in x & y, 0:1 in Z. Where Image space hidden surface algorithms work.
5. **Screen Space (2D):** Coordinates 0:width, 0:height

The 3D-pipeline

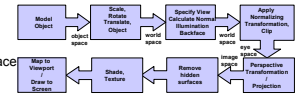
Example Image Space Viewing Algorithm Steps

1. Create World Space Scene:
   - Position each object (scale, rotate, translate)
   - Set up view parameters
2. Calculate Backface, illumination, etc.:
   - Can Perform Backface removal here
3. Calculate and Perform World to Eyespace transformation
4. Calculate and Perform Perspective Transformation
5. Clip Objects
6. Perspective Divide
   - Can Perform Backface removal here
7. Map to 3D Screen Space:
   - Viewport Transformation
8. Perform Hidden Surface Removal
9. Draw to Screen (Scan Convert):
   - Shade, texture

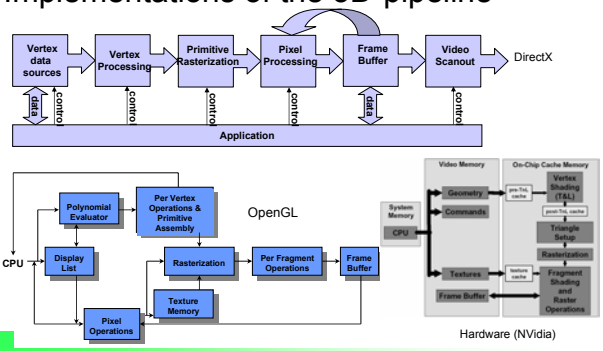


Steps in the 3D pipeline

1. Application/Scene
   - Scene/Geometry database traversal
   - Movement of objects, and aiming and movement of view camera
   - Animated movement of object models
   - Description of the contents of the 3D world
   - Object Visibility Check including possible Occlusion Culling
   - Select Level of Detail (LOD)
2. Geometry
   - Transforms (rotation, translation, scaling)
   - Transform from Model Space to World Space (Direct3D)
   - Transform from World Space to View Space
   - View Projection
   - Trivial Accept/Reject Culling
   - Back-Face Culling (can also be done later in Screen Space), Lighting
   - Perspective Divide - Transform to Clip Space
   - Clipping
   - Transform to Screen Space
3. Triangle Setup
   - Back-face Culling (or can be done in view space before lighting)
   - Slope/Delta Calculations
   - Scan-Line Conversion
4. Rendering / Rasterization
   - Shading
   - Texturing
   - Fog
   - Alpha Translucency Tests
   - Depth Buffering
   - Antialiasing (optional)
   - Display

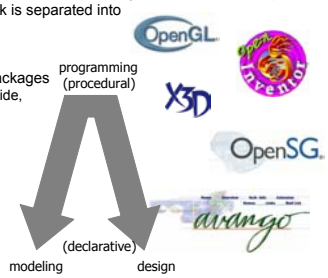


Implementations of the 3D-pipeline



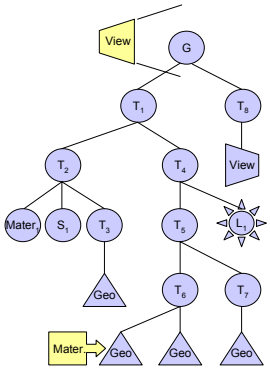
VR-development tools

- Several software packages and APIs support the real-time 3D-CG relevant processing stages.
- Complex tools support additional simulation tasks (auditory, tactile feedback, …).
- For mature simulations, the overall task is separated into
  - modeling the world and
  - designing the interaction.
- Direct rendering, gfx and simulation packages
  - OpenGL, Direct3D, GKS (3D), VCollide, OpenAL, ODE, Vortex, HAVOC, …
- Scene graph based tools and APIs
  - VRML, X3D, OpenGL Performer, OpenGL Optimizer, Open Inventor, OpenSG, PHIGS+, …
- VR modeling toolkits
  - AVANGO, World toolkit, Masive1-3, Dive, Lightning, game engines, …



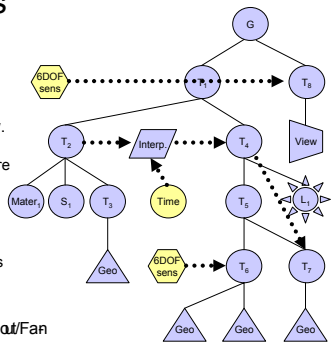
Scene Graphs

A Scene graph
- is a data structure for 3D-CG:
  - DAG – Directed Acyclic Graph or
  - Tree (no multiple instancing).
- organizes and controls the rendering of its constituent objects.
- consists of
  - a root node (base frame of reference).
  - intermediate nodes (grouping, transformation, sensors, engines, materials).
  - leaf nodes (geometry, lights, views).
- groups objects spatially to support
  - proximity detection.
  - collision detection.
  - view frustum culling.
  - occlusion culling.
- may support local or global state inheritance.
- may support different types of intermediate nodes



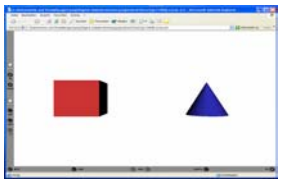
Application Graphs

An application graph
- is a data structure for 3D CG:
  - Can be either DAG or
  - DCG (Directed Cyclic Graph).
- organizes and controls data flow.
- constitutes a data flow network:
  - Node components (variables) are represented as **fields**.
  - Nodes are represented as **field containers**.
  - Fields of equal types can be interconnected by **routes**.
  - Changing field-values generates events processed
    - once per field change.
    - once per frame.
- can support multiple Field Fan out/Fan in for its fields.



A Scene Graph Language with Field Routing: VRML/X3D

```
#VRML V2.0 utf8
Transform {
 translation -3 0 0
 children Shape {
  geometry Box { }
  appearance Appearance {
   material Material { diffuseColor .8 .2 .2 } }
 }
}
Transform {
 translation 3 0 0
 children Shape {
  geometry Cone { }
  appearance Appearance {
   material Material { diffuseColor .2 .2 .8 } }
 }
}
```



More VRML later in this course!