Review of Tuesday

- Complex (mechanical) objects consist of components
- **Kinematic chains/skeleton** represented as tree
- Hierarchical modeling facilitates implementation of relative behavior of components

- **Relative** transformation matrices are used for animation
- Graphical representations can be reused

representation ‘attached’ to skeleton
Forward Kinematics

\[
p = T(dx_0, dy_0)T(0, dy_1)R(\alpha_1)T(0, dy_2)R(\alpha_2) \cdot \begin{pmatrix} 0 \\ dy_3 \end{pmatrix}
\]

positions of points in the mechanical system dependent on parameter values

\[
p = f(\theta)
\]
Forward and Inverse Kinematics

- **Forward Kinematics** computes the end-position of the kinematic chain dependent on parameter values

\[ p = f(\theta) \]

“Where is the hand of my robot if I choose these angles at the joints?”

- **Inverse Kinematics** finds parameters for a desired state/position

\[ \theta = g(p) \quad (\theta = f^{-1}(p) \quad (?)) \]

“What angles should I use at the joints to place my hand over there?”
Inverse Kinematics

\[ \theta = g(p) \quad (\theta = f^{-1}(p) \quad (?)) \]

- Parameters hard to find (f might not have an inverse)
  - No solutions ("Point cannot be reached by the robot’s hand.")
  - Multiple solutions ("Robot can reach the point in several ways.")

- In animation one tries to find an ‘optimal’ solution
  - Realistic, smooth movement
  - Minimal movement
  - Manual techniques (\textbf{key-framing}: Specify intermediate parameters)
Scene Graphs

- **Scenes** are similar to complex objects
  - They contain several (related objects)
  - Objects can be similar/share properties
  - Object may be grouped
  - May contain a logical or spatial hierarchy
  - Relative operations can be applied between related objects

- But: Camera is often located **in** the scene
Scene Graphs

- Scene Graphs are **object oriented** representations
- Enable **reusability** of components/objects (instancing)
- Hide the **state machine** character of OpenGL
Scene Graphs

- Scenes can be rendered by traversing the scene graph
- Attribute and matrix stacks

- Typical operations include:
  - Changing position/orientation/scale (geometric transformations)
  - Changing appearance
  - Changing shader/buffer states

- Different operations can be applied to the same object if it appears multiple times in the graph (multiple instances of the same object)
Scene Graphs

• Several **scene graph APIs** sit on top of OpenGL
  • OpenSceneGraph
  • Open Inventor

• Scene graph implementations
  • Make use of object oriented programming paradigms
  • Encapsulate OpenGL state changes
  • Ideally make it harder to write inefficient rendering code
  • Allow to render scenes as sets of independent objects
Spatial Hierarchies and Scene Graphs

- Scene graphs can correspond to **spatial hierarchies** in the scene

```
       House
      /     \
Room 1  Room 2
  /     /   /   \
Bed   Chair 1 Table Char 2
```
• Combining this hierarchy with spatial subdivision or spatial partitioning can greatly improve rendering performance
• This is because our camera can be located inside the scene.
Spatial Hierarchies

- **Spatial Partitioning** techniques can be combined with scene graphs to perform efficient visibility testing.

Passing the complete scene to the graphics card is inefficient, make use of spatial relationships and perform pre-selection/rejection.
Spatial Hierarchies

- Spatial Partitioning techniques can
  - Be combined with general scene graphs
  - Perform early rejection of objects
  - Reduce workload for graphics pipeline (transformation, clipping,…)
  - Reduce memory requirements (do not load invisible parts)
  - Adapt to scene properties/requirements
  - Be hierarchical in nature
Binary Space Partitioning

- **Binary Space-Partitioning Tree** (BSP-Tree)

  each plane in the scene performs a binary partitioning into two groups
Binary Space Partitioning

- BSP-Tree Construction
Binary Space Partitioning

- BSP-Tree Construction
Binary Space Partitioning

• BSP-Tree Construction
Binary Space Partitioning

- BSP-Tree Construction
Binary Space Partitioning

- BSP-Tree Construction

![BSP-Tree Diagram]
Binary Space Partitioning

- BSP-Tree Construction
Binary Space Partitioning

- BSP-Tree Construction
BSP Tree

• BSP-Tree Applications
  • Painter’s algorithm (sorting)
    • Planes sort objects into (“behind” and “in front”)
    • Tree traversal dependent on viewing position/orientation
  • Visibility tests
    • Tree computation is performed offline
    • Tree computation as pre-processing
    • Visibility information between nodes

• Drawbacks:
  • Increases number of polygons
  • Requires costly ordering and intersecting
  • Assumes static geometry
Quadtrees and Octrees

- **Quadtrees** subdivide space in a predefined way.

  - Parents have four child nodes.
  - Axis aligned separation planes.
  - Cell widths are halved.
Quadtrees and Octrees

- Quadtree example:
  - Image compression

- Octrees (in 3D) have eight child nodes per parent
Quadtrees and Octrees

- Quadtree example:
  - Hierarchical collision test
Scene Representation - Summary

- Scenes consist of multiple objects in the same frame
- Hierarchical representations (trees) are important for
  - **Compound/Mechanical objects**
    - Kinematic skeleton/chain represents relative transformations
  - **Scene composition** using a scene graph
    - Hierarchical, object-oriented representation
    - Object instancing, relative properties
    - Can be combined with spatial partitioning technique
  - **Spatial partitioning/Spatial subdivision**
    - Pre-computed visibility tests
    - Pre-computed relative ordering
    - Hierarchical representation (compression)
    - Hierarchical spatial operations (collision test)