3D space, transformations and scene organization

3D scene lives in a 3D space

- When it comes to defining shape, location, orientation and other properties of 3D scene elements we relay on using **points and vectors.**
- Points and vectors are represented with **three coordinates: (x,y,z)**
- These values are meaningless without a **coordinate system** that defines origin of the space and gives three linearly independent vectors that define x,y and z axis of a space.
- Origin and three vectors define a **frame** which defines coordinate systems.
- **Cartesian coordinate system.**
- <IMAGE: 3D COORDINATE SYSTEM, POINTS, VECTORS, OBJECTS>
  - Point or vector in 3D space depend on its relationship to the frame – point can have same absolute position in space but its coordinates depend on frame.

Other coordinate systems

- Spherical coordinates

3D scene requires a standard frame

- Note that coordinate system is defined with origin and three vectors.
- But point and vectors depend on coordinate system!
- Therefore, we define **world space** – a standard frame – with origin (0,0,0) and vectors (1,0,0), (0,1,0) and (0,0,1).
- All other frames will be defined with respect to this world coordinate system.

Note: coordinate system handedness

- Axis of x,z and z vectors defining coordinate system can face in one of two directions.
- Left coordinate system
- Right coordinate system
- <IMAGES>

Common geometry information

- Points
- Vectors
- Normals
- Rays
- Matrices
- Bounding boxes

Vectors

- Represent direction in 2D or 3D space
- Row or **column major**
- Two coordinates (x,y) – usually for texture space
- Three coordinates (x,y,z) – for any 3D elements
- Common operations: Addition, substracton, multiplication/division with scalar, etc.
- Dot and cross product operators
- Normalization
- Local coordinate system can be constructed using single vector using cross product.

Points

- Zero-dimensional **location** in 2D or 3D space: (x,y) and (x,y,z) with respect to coordinate system
- Although the same representation is used for vectors, point represents position while vector represents direction
- Substracting or adding point with vector results in new point
- Distance between two points results in vector which contains length and direction.
- Interpolation between points: linear interpolation

Normals

- Normal is a vector (x,y,z) perpendicular to surface at a particular position
- It can be defined as cross product of any two non-parallel tangent vectors at that point
- Similar to vectors but they are defined in relationship to a particular surface: they behave differently in some situations, particularly when **applying transformations**
- Also used for determining facing of the surface and hemisphere of that surface

Rays

- Line specified by its origin (point) and direction (vector)
- Often constructed by camera scene element
- Ray is parametrically expressed as a function of scalar value t:
  - $r(t) = o + t * d, 0 < t < inf$
- Ray differential: camera rays offset by one sample in x and y direction from main ray on film plane
  - Used for antialiasing of textures

Bounding boxes

- Acceleration of rendering often relies on subdividing the image or 3D scene into smaller elements and computing those either in parallel or ignoring if not needed.
- Examples of space subdivision:
  - Axis aligned boxes
  - Oriented bounding boxes
  - Bounding volume hierarchy

Matrices

- Matrices are essential for moving elements within 3D scene
- 2D array of numbers – row or **column major**
- Dimension is noted as m x n – number of rows x number of columns
- Square matrices: 3x3, 4x4
- Operators: multiplication, inverse, transpose, determinant
- Usage: multiplication with vectors and points → transformations

Matrix and coordinate system

- Relationship between matrix and coordinate system
- https://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/geometry/creating-an-orientation-matrix-or-local-coordinate-system

Transforms

- Transform is operation that takes points, vectors and converts them in some way.
- Basic tool for manipulating 3D scene elements:
  - Position, orientate, reshape and animate objects, lights and cameras
  - Essential for rendering computations
- Linear transforms: scaling, rotation
- Affine transforms: translation, rotation, scaling, reflection, shearing
  - Preserve parallelism of lines but not necessary lengths and angles
  - Homogeneous notation

Transforms in modeling

- https://www.youtube.com/watch?v=Hf2esGA7vCc
-

Transforms in animation

- https://www.youtube.com/watch?v=CBJp82tlR3M
-

Transforms in rendering

- Ensure that all calculations are preformed in the same coordinate system
- An important example is during light-surface calculation
- Another example is projecting objects onto plane which is used for rasterier-based rendering.

Transforms in professional software

- Godot: https://docs.godotengine.org/en/stable/classes/class_transform.html
- Unity: https://docs.unity3d.com/ScriptReference/Transform.html
- Unreal: https://docs.unrealengine.com/4.27/en-US/BlueprintAPI/Math/Transform/
- Blender: https://docs.blender.org/manual/en/latest/scene_layout/object/properties/transforms.html
- GLM: https://github.com/g-truc/glm

Basic transformation matrices

- Modeling elements, 3D scene and animations relies on transformations
  - Translation
  - Rotation
  - Scale
  - Shear
  - Look-at notation

Translation matrix

- Used to change point from one location to another
- <matrix>
- Note that vector is not affected by translation matrix – direction can not be translation.
- Inverse: $T^{-1}(t) = T(-t)$
- Rigid-body transform: preserves distances between points and headedness

Rotation matrix

- Rotates vector/point by given angle around given axis passing through origin:
  - 3 matrices: rotation of theta degrees around x, y, or z axis can be used for rotation around arbitrary axis
- Pivot point is required for determining rotation: object is first translated so that pivot point in the center and then rotation is performed after which object is again transformed so that pivot point is in the same position as before.
- rigid-body transform: preserve distance between points and headedness
- useful for orienting objects: e.g., orientation matrix - rotation matrix associated with camera view or object that defines its orientation in space: directions for up and forward are needed to define this matrix.
- Inverse: $R^{-1}(phi) = R(-phi)$
- Determinant = 1, orthogonal

<Example image>

Scaling

- Enlarging or diminishing objects
- If all scaling factors are same: uniform (isotropic) scaling, else non-uniform (anisotropic).
- Negative value of scaling factor gives **reflection (mirror) matrix**
  - Triangle with clockwise orientation will have counter-clockwise orientation after reflection

<Example image>

Shear matrix

- 6 basic shearing matrices
- TODO

<Example image>

Practical note: concatenation of transforms

- Matrix multiplication is noncommutative: order of multiplication matters
- Concatenating scaling, rotation and translation must be made in this order: T * R * S
  - Scaling is applied first, then rotation and finally translation.
- Concatenation of only translation and rotation matrices results in rigid-body transform.
  - Inverse

<Example image>

Practical note: moving directed objects

- Camera movement using **look-at notation**
- **TODO**

<Example image>

Special care: transformations on normals

- TODO

<Example image>

Special transforms

- Euler transform
- Rotation about an arbitrary axis
- Quaternions

Euler transforms

- Euler matrix is concatenation of 3 rotation matrices around x,y and z axis
- Requires establish default view:
  - negative z
  - head orient: along y
- Angles of rotation: head (y-roll), pitch (x-roll), roll (z-roll)
- Problems:
  - gimbal lock,
  - two different sets of Euler angles can give same orientation and finally interpolation between two Euler angle sets is not as simple as interpolating each angle
- Good: angles can be extracted from Euler matrix -> matrix decomposition

<Example image>

Rotation about arbitrary axis

- Transform to space where axis we want to rotate around is x, perform rotation, return to original space.

<Example image>

Quaternions

- Represent rotations and orientations
- 3D orientation can be represented as single rotation around particular axis
- Advantage: Interpolating between two quaternions is stable and constant
- Init quaternion represents any 3D rotation

<Example image>

Projections

- Perspective
- Orthographic
- TODO

<Example image>

Scene graph

3D scene as scene-graph

- 3D scene modeling goes hand in hand with object oriented design.
- 3D scene representation has inherent tree-like structure thus often represented with so called **scene-graph**
  - Arrangement between user who builds 3D scene and renderer
  - https://learnopengl.com/Guest-Articles/2021/Scene/Scene-Graph
  - Scene modeling tools: DCC examples

<IMAGE: COMPONENTS OF 3D SCENE AND SCENE GRAPH>

Scene graph

- Hierarhical datastructure for organizing and structuring storing whole 3D scene, with all its elements
- User oriented datastructure for modeling and organizing elements and their relationships in hierarchy for easier manipulation and construction of scene
  - It is edited and created by user: artists and designers
  - Enables easier modeling and animation (e.g., whole subtrees can be translated)
- It serves as support to rendering algorithms. Pass through scene graphs pass for rendering
  - Depending on rules, rendering algorithm might use different materials, geometries, lights, animations, etc.

Scene graph

- Example of simple scene graph

Scene graph

- Elements:
  - Nodes: root, internal and leaf. Each node contains data, at least its position in graph which gives consistent structure
  - Edges
- Graph scene:
  - Root – starting point for whole scene
    - Data: type of coordinate system, scene units, etc.
  - Internal nodes – organize scene into hierarchy. Often those are transformation information (where and how are objects positioned)
  - Leaf nodes – contain elements of the scene: objects, camera, lights. As objects in the scene can repeat, these nodes can be duplicated
  - image

Leaf nodes

- Contain elements of the scene:
  - Object meshes
  - Object materials
  - Lights
  - Cameras
  - EXAMPLES

Internal nodes

- Often those are transformation nodes defining positions of sub-trees: translations, rotations, scaling, etc.
- Other types:
  - Grouping – contain nodes without any other function
  - Conditional – type of grouping node which enables activation only the particular children node
  - Level of detail – contains children nodes where each has a copy of objects with varying level of detail and only one is active depending on camera distance
  - Billboard – groupping node which orientates all children node towards camera
  - EXAMPLES
- As materials and textures are often shared between different objects, they can be stored as internal node which is references by children nodes

Scene graph traversal

- Recursive operation starting from root going through hierarchy
- Scene graph traversal can be used during rendering
- Therefore, it serves as **standardized scene description** that can be shared between different rendering, modeling and interaction tools

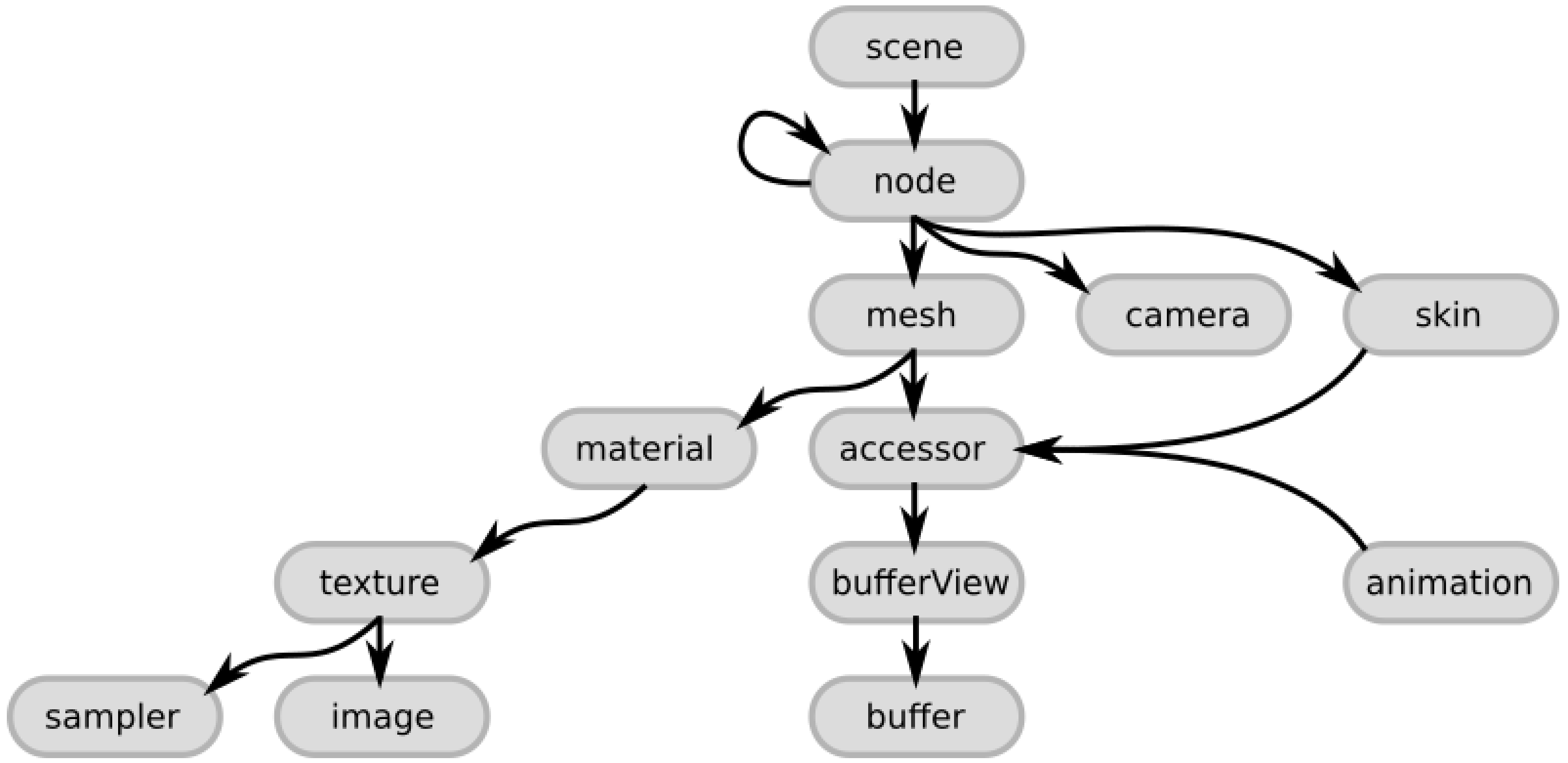
Scene graph and scene transfer

- Different modeling, rendering and interaction software has different formats for scene description
  - Transfer between them requires **standardized formats**
  - For different formats, different **importer/exporter** functions are needed
  - Different formats exists which differ by **scene elements support**
- Storing and transferring scene requires data described by scene graph

Scene graph and scene transfer

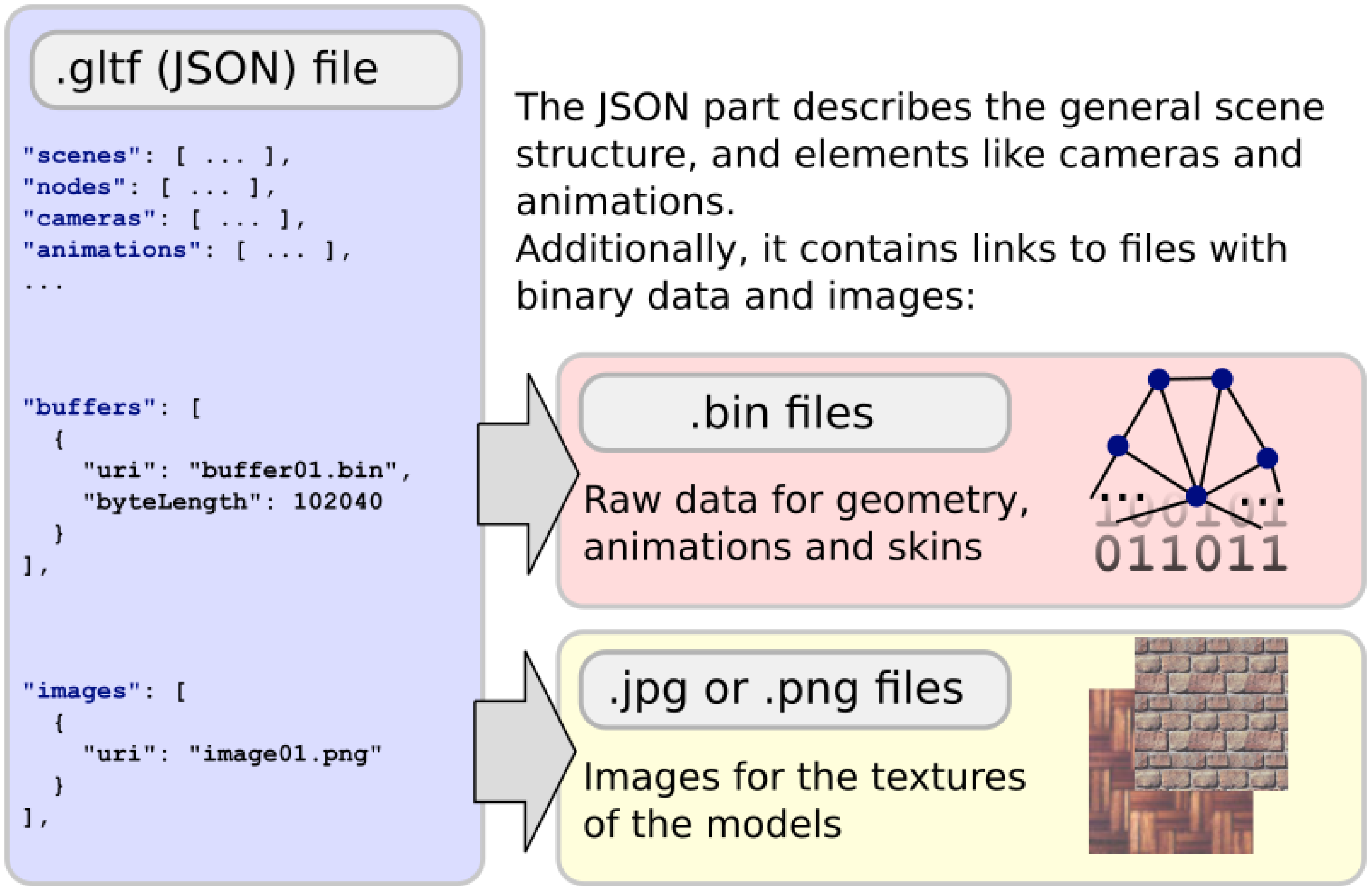
- Tendency towards standardized formats is required
- Popular scene description formats:
  - glTF
  - USD: https://graphics.pixar.com/usd/release/index.html

Scene graph example: glTF



https://github.com/KhronosGroup/glTF-Tutorials/blob/master/gltfTutorial/gltfTutorial_002_BasicGltfStructure.md

Scene graph example: glTF



```
"scenes": [ ... ],
"nodes": [ ... ],
"cameras": [ ... ],
"animations": [ ... ],
...


"buffers": [
  {
    "uri": "buffer01.bin",
    "byteLength": 102040
  }
],


"images": [
  {
    "uri": "image01.png"
  }
],
```

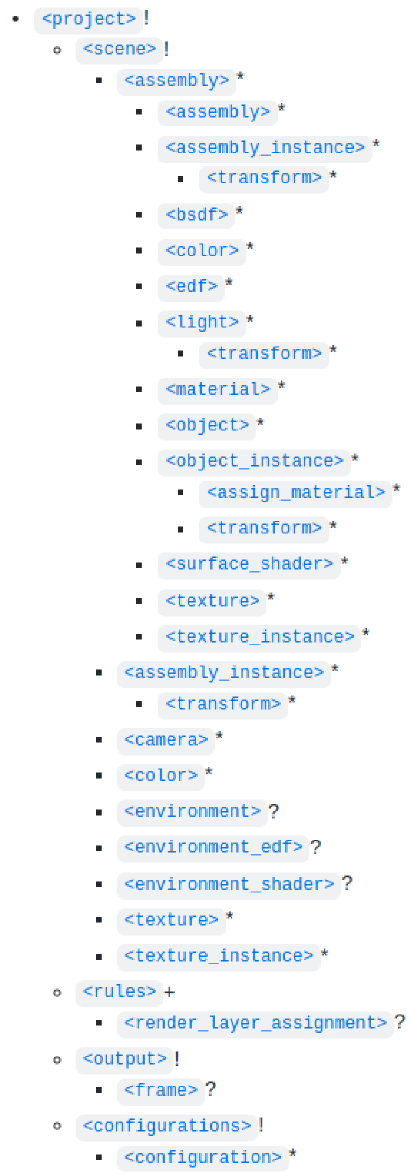
The JSON part describes the general scene structure, and elements like cameras and animations.
Additionally, it contains links to files with binary data and images:

Raw data for geometry, animations and skins

Images for the textures of the models

Elements of 3D scene in production

- https://github.com/appleseedhq/appleseed/wiki/Project-File-Format
- Scene graph can be stored in various formats: e.g., XML



- `<project>` !
  - `<scene>` !
    - `<assembly>` *
      - `<assembly>` *
      - `<assembly_instance>` *
        - `<transform>` *
      - `<bsdf>` *
      - `<color>` *
      - `<edf>` *
      - `<light>` *
        - `<transform>` *
      - `<material>` *
      - `<object>` *
      - `<object_instance>` *
        - `<assign_material>` *
        - `<transform>` *
      - `<surface_shader>` *
      - `<texture>` *
      - `<texture_instance>` *
    - `<assembly_instance>` *
      - `<transform>` *
    - `<camera>` *
    - `<color>` *
    - `<environment>` ?
    - `<environment_edf>` ?
    - `<environment_shader>` ?
    - `<texture>` *
    - `<texture_instance>` *
  - `<rules>` +
    - `<render_layer_assignment>` ?
  - `<output>` !
    - `<frame>` ?
  - `<configurations>` !
    - `<configuration>` *

Literature

- https://github.com/lorentzo/IntroductionToComputerGraphics/wiki/Foundations-of-3D-scene-modeling