More on 3D scene

Scene organization

- 3D scene is composed of objects, lights and cameras.
  - During rendering, those elements are employed for shading and light transport to obtain the image.
- Generally, 3D scenes are very complex. They often contain a large number of lights, objects and cameras
  - Example of complex scene which is actually normal in everyday graphics
- Although computing power is large and largely increased from 10, 20, 30 years, the goals in rendering were also advancing:
  - More frames per second (real-time applications, 60-90 fps)
  - Higher resolutions with more samples per pixel (4K, 8K screens)
  - More realistic materials, lights, shapes, cameras, shading and light transport (millions of polygons, physically based materials, high-resolution textures, advanced camera simulations, photo-realistic push, etc.)

Scene organization

- More complex scenes + more advanced computing power = same rendering time (find the law, Blinn?)
- Even efficient rendering algorithms can not handle the complexity of scenes alone. Acceleration datastructures will always be needed
- To perform efficient rendering and simulation, on application phase (phase prior rendering) data-structures for efficient scene organization exist.
  - Such datastructures enable faster check of objects which are not visible from camera – remember that naive rendering implementation checks all objects in the scene for visibility although a small portion of the scene is visible from camera.
  - Another example is faster check of relative positions between objects in the scene needed for animation (e.g., rigid body simulation)

Scene organization

- Three main data structures:
  - **Spatial datastructures**: group elements based on relative distances for fast check of their relationships or position in space
  - **Scene graph**: model and organize elements relationships in hierarchy for easier manipulation and construction of scene
  - State partitioning: group elements based on their characteristics for faster rendering

Spatial datastructures

- Organize geometry in 2 od 3 dimensional space
- Accelerate queries for geometric overlap
- Used for: culling, intersection testing, ray-tracing and collision detection
- Such datastructures are hierarhical, nested and recursive
    - Each node has children which define volume of space
    - They give improvement of complexity from O(n) to O(log(n)) – instead of searching through all n objects, we only look at subset of those
    - Construction of such datastructures might take time but it is more efficient in the long run for rendering*

* This is highly researched field constantly pushing speed of datastructures construction so that they can be constructed in real time.

Spatial datastructures

- Common types:
  - BVH
  - BSP
  - Quad/oct-trees

BVH

- Not a space subdivision structure, it encloses regions of space surrounding geometrical objects and thus not enclosing whole space
- TODO RTR 19

BSP

- Based on space subdivisions
  - Entire space is subdivided and encoded in datastructure: union of all leaf nodes is equal to entire space
  - Normally, volumes of leaf nodes do not overlap
- Different variants exist and they can be implemented to arbitrary subdivide the space
- TODO RTR 19

Octrees

- Based on space subdivisions
  - Entire space is subdivided and encoded in datastructure: union of all leaf nodes is equal to entire space
- Octrees uniformly divide space which can be source of efficiency
- Normally, volumes of leaf nodes do not overlap, but loose octrees can have overlapping volumes.
- TODO RTR 19

Scene graph

- BVH, BSP and octrees:
  - Partition space
  - Store geometry and nothing else
- 3D scene contains much more than just geometry: lights, cameras, materials, other shape representations
- Scene graph:
  - User oriented datastructure (node hierarchy in glTF)
  - Stores: textures, transformations, levels of detail, render states (e.g., material properties), light sources, etc.
  - Control of animation and visibility
  - Represented by a tree
  - Can contain spatial datastructures

Scene graph examples

- Example: light can be one node which affects only the contents of subtree
  - Image
- Example: one material can be applied to multiple objects
  - Image
- Example: level of detail RTR figure 19.34

Optimizations

- Level of detail
- Culling
- TODO RTR 19

Further into topic

- Rendering large scenes: RTR 19

More topics in 3D scene modeling

- Animation
- Interaction
- Complex shape modeling
- Complex material modeling
- More on lights
- More on cameras

3D scene: Animation

Animation

- Introducing time component
- Types:
  - Environment: phenomena and effects
  - Character: face and body animation
- Approaches:
  - Manual
  - Procedural
    - Phenomenological models
    - Physics simulation

Animation tools

- Particles
- Meshs: deformation of vertices
  - Blending and morphing (RTR 4.4, 4.5)
- Voxels
- Splines
- Interpolation

Manual animation

- Rigging and bones
- Interpolation

Motion capture

Procedural animation

- Physics:
  - Fluid simulation
  - Static body and collision
  - Rigid body and collision
  - Kinematics and inverse kinematics
  - Cloth, hair

Texture animation

- Image applies to surface can be dynamic
- Texture coordinates do not have to be static
- Phyiscal simulations or procedural textures can be stored in array of images
  - Example: VFX using Houdini

3D scene: Interaction

Interaction

- HCI

- 3D scene: Alternative shape representations

Implicit surface: basics

- In foundations of 3D scene modeling we have discussed parametric curves and surfaces.
- Implicit surfaces form another useful class for modeling and **shape representation**.
  - Often used in **intersection testing with rays** since they are simpler to intersect than parametric surface
- **Implicit function**: f(x,y,z) = 0
  - Point (x,y,z) is on implicit surface if the result is zero when point (x,y,z) is inserted into function
  - Signed distance: negative if (x,y,z) inside or positive if (x,y,z) outside will be returned – therefore name **signed distance functions** is used**.**
  - Normal (essential shading information) can be computed using partial derivatives – the gradient of f. <FORMULA>
    - In Practice, central difference can be used for approximating gradient <FORMULA>
- <EXAMPLES of SDFs: https://iquilezles.org/articles/distfunctions/>

Implicit surface: modeling

- **Constructive solid geometry** algorithms (subtraction, addition - union) can be easily done with them
- SDFs can be easily deformed or blended
  - This is called blobby modelling → **metaballs**
- <examples>:
  - https://www.playstation.com/de-de/games/dreams/
  - Blender metaballs
- Transformations are done using the inverse transform applied to p, e.g., f(p-t)
- Repeating object can be done using r=mod(p,c).

Implicit surface: rendering

- For visualization, ray-marching is often used.
  - This method also enables calculating shadows, reflections, AO and other effects.
  - <example: raymarching>

Implicit surface: rendering

- Another approach is to turn the SDF into surface consisting of triangles.
- Famous algorithm for this is **marching cubes algorithm***

* Different polygonizational algorithms and GPU methods are presented for real-time rendering.

Isosurface: examples

- https://dl.acm.org/doi/10.1145/3023368.3023377
- **https://store.steampowered.com/app/661920/Claybook/**
  -

Subdivision curves and surfaces

- Used to make **smooth** curves and surfaces
- Gap between discrete surface (triangle meshes) and continuous surfaces (e.g., Bezier patches)
  - Useful for dynamic level of detail

<example: mesh surface, parametric surface, subdivision surface>

<example:level of detail>

Subdivision curves: intuition

- Corner cutting algorithm explains subdivision curves well.
  - Initial polygon P_0 is given which specifies control polygon: vertices are control points
  - Corners of given polygon are cut and this is repeated until **sharp corners are removed** $P_0 \rightarrow P_1 \rightarrow \ldots \rightarrow P_n$
  - Result is called **limit curve** → smooth curve since all corners are cut off
  - Analogy: low-pass filtering – all sharp corners (high frequency) are removed.
  - <image>

Subdivision curves

- Subdivision process can be done in many possible ways
- Subdivision surface is characterized by subdivision scheme
- <example: meshlab subdivision> https://pymeshlab.readthedocs.io/en/0.1.8/filter_list.html#subdivision_surfaces_butterfly_subdivision
-

Subdivision curves: Chaikin scheme

- Given initial polygon P0 with n vertices this is simple method to raptly generate smooth curve
- Chaikin scheme creates two new vertices between each subsequent pairs of vertices
- After each step, original vertices are discarded and new points are reconnected – new points are created ¼ away from original vertices
  - <image>
  - <formula>
- Properties:
  - This scheme generates quadratic B-spline
  - Works only for closed polygons

Subdivision curves: schemes

- Two main subdivision schemes exists:
  - Approximating
  - Interpolating
- Approximating
  - Chaikin scheme
  - Since (original) vertices are discarded or modified, limit curve does not lie on vertices of original polygon
- Interpolating
  - Contains all vertices from previous step: limit curve goes through all points - interpolating

Subdivision curves: interpolating scheme

- An example: 4 point interpolatory subdivision scheme*
  - 4 nearest points are took for creating a new point
  - All points from previous step are kept
  - Tensions parameter: 0 → linear interpolation, > 0 → curves with different continuity parameters
  - Does not work directly for open polygons.
  - <formula>
  - <image>

* https://cgvr.cs.uni-bremen.de/papers/c2scheme/c2scheme.pdf

Subdivision curves: in practice

- Examples in animation and modeling: http://multires.caltech.edu/pubs/sig00notes.pdf
-

Subdivision surfaces

- Introduced concepts on subdivision curves apply to subdivision surfaces
- Paradigm for defining smooth, continuous surfaces from meshes with arbitrary topology
  - Infinite level of detail is provided: arbitrary number of triangles/polygons can be generated
  - <example>
- Inspection of continuity is mathematically involved process and thus not straightforward.
- Simple and easily implemented rules needed for subdivision. Two phases, that characterize subdivision scheme:
  - In first, **refinement phase**, new vertices are created and reconnected for some or all vertices of **control mesh** (initial mesh). Different refinement methods exist (e.g., how the polygon can be split)
  - Second, **smoothing phase**, computes new positions for some or all vertices in the mesh. Different schemes dictates the continuity of surface and whenever the surface is approximating or interpolating.
  - <phases example>

Subdivision surfaces: types

- Subdivision schemes can be:
  - Stationary or non-stationary
    - Stationary – same rules are used in each step
    - Non-stationary – changes steps based on current step
  - Uniform or non-uniform
    - Uniform – same rules for every vertex or edge
    - Non-uniform – different rules for vertices or edges, e.g., edges that are on boundary of a surface.
  - Triangle or polygon based

Subdivision surfaces: Loop subdivision

- TODO RTR BOOK
- Meshlab example

Subdivision surfaces: Catmull-Clark

- TODO RTR BOOK
- Meshlab example

Displacement and subdivision

- TODO RTR BOOK
- Blender example

Subdivision surfaces: production

- Pixar OpenSubdiv
- RTR BOOK

Subdivision surfaces

- Catmull-Clark subdivision surfaces

Operations on meshes

- Tessellation and triangulation – RTR 16
- Consolidation – RTR 16
- Simplification – RTR 16
- Compression – RTR 16

Constructive solid geometry

- Two main methods for visualization:
  - Stencil buffer
  - Analytical storage

Voxels

- TODO

3D scene: digitalization

Digitalization

- Scanning, photogrammertry, measuring

3D scene: procedural modeling

Procedural modeling

- Shape
  - Procedural geometry
- Material
- Noise
- Shaders
- Noding system

3D scene: more on material and textures

Complex material modeling

- Physically based scattering functions
- PBR textures

Only to know that it exists

- Wave optics BRDF models
    - TODO

Enhancing texture representation

- When handling many textures in application there are several enhancements for improving performance available:
  - Texture compression
  - Texture atlases, arrays and bindless textures

Texture atlas

- For efficiency reasons, it is good to batch up as much as possible work for GPU and change states as little as possible
- To do so, texture alas can be used: putting several images (subtextures) into single larger image
  - Example
- Optimization of image texture as well as their mipmaps is imporant for efficient storage and retreval
- Problems:
  - Wrapping/repeating and mirror modes affect the whole texture rather than subtextures
  - Mipmapping must be done before creating the atlas otherwise colorbleeding might occur

Texture atlas in production

- Ptex
  - System wheere each quad in subdivision surface has its own small texture
  - This approach doesn’t require unique texture coordinates over mesh and thus no artifacts over seams of disconnected parts of texture atlas are present
  - Widely used in animation: for painting texture on 3D models directly
  - <example>
- Many other methods build on this one
  - Packed Ptex
  - Mesh color textures
  - HTex

Texture array: API note

- Graphics library API function
- Subtextures in array must have same dimensions, format, mipmap heirarchy and MSAA settings
- Avoids problems with mipmapping and repeat modes present in texture atlas

Bindless texture: API note

- With bindless textures there is no upper limit on the number of textures

Texture compression

- RTR 6.2.6

Parallax mapping

- Looking at brick wall, we can see that under some angle we wouldn’t see mortar between bricks
- This is not possible with normal/bump mapping since only normal is changed and no real geometry is generated.
  - Bumps never shift location with view location
  - Nor bumps block each other
- **Parallax** - positions of objects move relative to one another as the observer moves.
- Key idea of parallax mapping is to approximate what should be seen in a pixel by examining the height of what was found before.
- TODO

Displacement mapping

3D scene: light

More on lights

- Environment: sun and sky
- Area lights and shadow
- Light distributions

Environment illumination: cube map

- Cube map is accessed with three-component texture coordinate vector that specifies direction of ray pointing from center or cube to outward.
- Environment mapping is done with cube map so that HDRI image is used as faces of the cube.
- Camera is always in the center of the cube so that environment doesn’t move as camera moves.
- <example>

Textured lights

- RTR 6.9

IES profiles

- TODO

Shadows

- RTR 7

3D scene: Camera

More on cameras

- DOF and lenses
- Camera effects
- Motion blur
- exposure