Radiometry and Photometry

Flux: $\phi \equiv \frac{dQ}{dt}$, Solid Angle: $\omega \equiv \frac{A}{r^2}$, Intensity: $I(\omega) \equiv \frac{d\phi}{d\omega}$, Surface Irradiance: $E \equiv \frac{d\phi}{dA}$, Surface radiance: $L \equiv \frac{d\phi}{dAcos\theta d\omega}$

For Isotropic Point source: $\phi \equiv 4\pi I$

Lambert's cosine law: $E \equiv \frac{\phi}{A} cos\theta$, where $cos\theta \equiv l \cdot n$, n is surface normal

Evtl noch Lightning calculations

BRDF and Materials

Reflection types: Specular(mirror), diffuse(matt), gloss(mirror with deviation), retro(reflect in direction of source)

Bidirection reflectance distribution function (BRDF) defines reflection behaviour. $f(\omega_i, \omega_o)$ defines how much light from given in vector goes to given out vector

Properties: Linearity, Reciprocity, Isotropc, Anisotropic, Energy conservation

Reflection Equation:

$L_r(\omega_r) \equiv \int_{\omega_i} L_i(\omega_i) f(\omega_i, \omega_o)(\omega_i \cdot n) d\omega_i$

Energy Conservation: Incoming Radiance has to be greater or equal to outgoing Radiance: $\int f_o(\omega_i, \omega_o) cos\theta_o d\omega_o \leq 1$

Example Lambertian: $f = \frac{\rho}{\pi}$

Surface appears equally bright from all directions (albedo: $\rho \leq 1$)

roughness of surface depends on surface normals →can be approached with gaussian distribution

Ray Tracing Methods

FW RT: Start from Light Source

BW RT: Start from eye(less comp.)

Recursive RT: Create Ray through pixel. At Intersection Point create 3 new Rays. 1.Shadow(compute with BRDF), 2.Reflected(in=outangle), 3.Transmitted(depends on mat, $\gamma_i sin\theta_i \equiv \gamma_o sin\theta_o$, if $\theta_i > \theta_o$ no transmission

Calc Shadow Intensity directly from LS, Reflection Intensity and Transmission Intensity recursively. Final Intensity at pixel is sum of these 3.

Scale Reflection and Transmission with k_r, k_t

Can visualise recursion with Ray Tree, stop at certain depth or when contribution is neglible

Ray Tracing Acceleration & Obj intersection

Ray: $r(t) = p_0 + d * t$, in Equation einsetzen und 0 setzen - Evtl formeln added für spezielle geom. objekte, bzw. allgemein geometry

Acceleration: bind complex obj's to simple shape and check shapes first when calculation obj-ray intersection

2 Methods: spatial(KD-Tree) or Obj partition(BHV)

KD-Tree: internal node = area split, leaf node = area with obj references

Let ray go through & check which areas are hit

Bounding Volume Hierarchy: Partition obj in subsets & create box around them, internal node: boxes, leaf node = obj references of the box

Anti-aliasing methods in RT: supersampling, adaptive sampling(supersamling in areas with a lot of change), stochastic sampling

Stoch. sampling methods: N-rooks(large grid with no 2 pieces in same row&col), Jittered: small grid with random point in each field, poisson: random & retry if too close to prev points

Distributed Ray Tracing, Monte Carlo Integration

Distributed/Stochastic Ray Tracing: enable visual effects by sampling multiple rays at all stages(eye, reflection, shadow, transmission)

eye rays: no jaggy edges(bokeh)

reflection: glossy surface

shadow: smooth shadow

transmission: translucency

rays over time: motion blur

use samples to estimate a numerical integration for the exact Intensity of pixel

Monte Carlo integration:$X_i \sim p(x)$, $F_N = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i}{p(X_i)}$

$p(X_i)$ represents importance of given sample and should approximate function -> Importance sampling

Use Monte Carlo integration for BRDF estimation

Sample with previous methods to avoid empty holes

Global Illumination and Rendering Equation

Global illumination: direct & indirect light(multiple bounces)

Rendering equation: Radiance depend on incoming radiance, incoming radiance can be rewritten as outgoing of another point(transport function!). Gives us Recursion

$L_o(p, \omega_o) = L_e(direct) +$

$\int_{\omega_i} f(p, \omega_i, \omega_o) L_o(tr(p, \omega_i), -\omega_i) cos\theta_i d\omega_i$

Simplified: $L_o = L_e + R(T(L_o))$, where $R(L_i)$ is reflected intensity, and $T(L_o)$ defines gives us the reflection point $K = R(T())$ which gives us 1 bounce (use it multiple times!)

Path tracing: Sum over all paths, terminate low impact paths (based on BRDF), Partition recursive radiance evaluation (direct & indirect part), monte carlo estimate for each partition seperately

Irradiance Caching: Store indirect illumination of nearby values&interpolate -> safes computation

Photon Mapping: tracing rays from eye is inefficient esp. for transparent objs, cast ray from light source & store on surface, then perform gathering

Curves and Surfaces

Limits of polygonal mesh: planar facets, fixed res, deformation is difficult, no natural math. parameterization

We want a smooth curve/surface given certain control points (interpolation

DeCasteljau Algostep: $p_{new}(u) = (1-u)p_i + up_{i+1}$, repeat until left with 1 control point (lies on bezier curve), do it for $u \in [0, 1]$

Piecewise bezier curve: Split control points into two groups, calc bezier curve for each and connect them.

Bezier Surface: 3d points given, group them into k groups. Then for $(u, v) \in [0, 1]^2$, do: Calc DeCasteljau for each group with u. Use resulting k points and calculate deCasteljau with v. Resulting point lies on surface. (classic 4x4)

Subdivison Surfaces

Mesh upsamling → increase resolution via subdivision
Mesh Downsampling → Decrease Resolution, try to preserve shape
Mesh Resampling → modify distribution to improve quality
Subdivison Surface: Start with coarse polygon mesh, subdivide each element, update vertices via local averaging
Loop subdivision: Split Triangle into four, assign new vertex position according to weight, update old vertices
Catmull-Clark Subdivision: split each face into multiple faces, compute center of each face, compute midpoint between each center, compute the new coordinate of old vertex: $(4 * avg_{mid} - 1 * avg_{center} + (n-3) * old)/n$, where $n = \#faces$ the vertex belongs to.
Mesh simplification via quadric error(iteratively remove an edge with the least impact)
Mesh regularization: try make trianlges uniform in shape and size

Volumetric Rendering

Volumetric Rendering: cell = volume between eight grid points
tri-linear interpolation: $(1-a)(1-b)(1-c)u_{000} + ... + abcu_{111}$
Techniques:
- Surface Rendering: indirect, convert volume into surface repr. & render them with conventional methods, marching cube algo
- Direct Voume Rendering: directly deal with 3d volume, volumetric ray casting
iso-surface = F(x,y,z), function to evaluate relative pos to surface
Marching cube: method to extract iso-surface from 3d volume
for each vertex check F(x,y,z) value → interpolate vertex for each cell depending on which vertex lie inside struct(256 cases → 15 base cases + rotation)→ use LUT do decide which triangles to render, then interpolate trianlge vertex pos based on F(x,y,z) of vertices & tri-linear interpolation
Volumetric ray casting idea: Cast rays through data volume to accumulate sample colors from front to back. Then render volume directly
- To estimate the color of a voxel(3dpixel) we need to estimate the normal vector at each voxel first: $g_x(i,j,k) = \frac{f(i+1,j,k) - f(i-1,j,k)}{2D}$ //do it for y, z with +1 at the respective position
ray-box intersection as previously discussed
color accumulation front to back: $C = a1C1 + a2C2(1 - a1)$... a is opacity of voxel

Computer Animation

12 Animation principles: Squash&Stretch, Anticipation, Staging, Straight ahead/pose-to-pose, follow through, ease-in and ease-out, arcs, secondary action, timing, exaggeration, sold drawings, appeal
Keyframe interpolation: Computer generates interpolated frames between Keyframes
Character animation techniques:
Skinning: transform each vertex with each bone rigidly & then blend using weights Rigging: skeleton of bones, apply trafo to each & move surface accordingly to bone
Forward Kinematics(FK): Describe position of body parts as function of joint angles
Inverse Kinematics(IK): Compute joint parameter along the chain to match desired position by user
Motion Capture: record real-world performance, extract pose from data, via optical, magnetic or mechanical
Crowd Simulation: Agent-based(individual, comput. heavy), Global(integrate global path, collision avoidance)
Physical Simulation via Particle System(Diff eq. to update pos over time, dynamics where we consider forces on particle), Finite Element Methods(FEM)(num method for finding approx. sols)

AR/VR

VR(User immersed in virtual world), AR(Real world expanded), MR(mixed)
Important topics: display, rendering, tracking
Display: 360 degree FOV, different perspective for each eye(stereo), user-motion(change view as user moves)
human eye's have only a small fraction with high resolution → track eye's and render specific regions with higher res to compute efficiently
Rendering: achieve presence(illusion of reality), needs very low latency
Foveated rendering: low resolution can lead to aliasing/flickering be careful, "tunnel vision" = blur periphery include lens distortion, we may then compute shading, that won't be used -> Lens Matched Shading (use lens distortion info in shading)
Tracking: Low-latency, 6DoF head motion, track gaze, eyes, camera pos, hand, finger ... for interactions
6Dof head pose estimation: 3D pos and 3D rotation, given rel. 3d pos of marker, camera vp, 2d pos of headset in im → solve for unknowns
Inside out tracking: multiple wide-angle cameras in front of headset, reconstructs world & tracks hands ...
Hand tracking: Glove, marker-based, depth camera
eye tracking: point of gaze, motion of eye related to head, or with infrared LED & camera