Image-based Rendering (IBR) (Vorlesung 19c)

"How can we render images without time-consuming computation?"

Plenoptic Function:

- theoretical basis to acquire data
- $p = P(\underbrace{\lambda}_{\text{wave-length}}, \underbrace{V_x, V_y, V_z}_{\text{point-coordinates}}, \underbrace{\theta, \phi}_{\text{all directions}}, \underbrace{t}_{\text{every time}})$ → 7D-function
- it describes all the image informations and stores the resulting radiance
- "How do we acquire this function?"
  → we need to discretize and reduce the dimensions
- Reducing the dimensions
  5D: Reduced Plenoptic Function (time and wavelength fixed)
  4D: Light Field Rendering (viewpoint is outside of objects)
  3D: Concentric Mosaics (viewpoint lies in a circle)
  2D: Panorama (fixed viewpoint)

Panorama (2D):

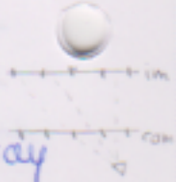
projection surfaces

- requirements: fixed viewpoint with special panorama cameras (rotating, mirrored, multiple cameras)
- how to do this: · you register multiple images (acquisition)
  · stitch them together and warp into one env. (mosaicing)
  · then resampling and blending (rendering)
- Mosaicing: first prewarping (cancel the distortion in the camera),
  then image registration (aligning),
  then composition (removing moving objects)
  at last resampling (filling holes, blending, filtering)
- Projection surfaces: on a ~~sfere~~ (inside) a cylinder, a cube (!distortion)

Light Fields (4D):

- reduced plenoptic function → object is inside bounding box and can't be penetrated
- you can generate new views and even view-depending lighting effects (see BRDF) (vgl. parrallax phenomena)
- Aquisition: · you have a static 3D scene on image plane
  · then take an image for each pixel on cam. plane
  → slightly different angles of same scene
  → 2D matrix of 2D images: Light field

- Rendering: · position camera somewhere behind camera plane (arbitrary viewpoint & view. dir)
  · shoot rays through image matrix or also called camera plane
  · then compute intersections with each plane
  · option 1: take the image on the camera plane which is the closest to the ray
  · option 2: · interpolate bilinearly on camera plane
    · interpolate bilinearly on image plane
    · combine the two and interpolate quadrilinearly



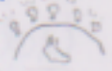
→ to generate aliasing-free images you need a lot of images (to generate photorealistic rendering results) (number of images → ∞ image resolution) !

- Parameterization: · we used two plane parameterization
  · also possible: point-angle, two points on sphere, original images & camera positions
- acquisition: computer-controlled camera rig, spherical camera motion
- advantages: · rendering cost is completly independent from scene complexity or material properties
  · also looks very realistic
- disadvantages: · can't change arrangement of objects or lights (static geometry and fixed lighting)
  · high storage costs to avoid aliasing

Reflectance Field

"How can we use light fields but with varying lighting?"

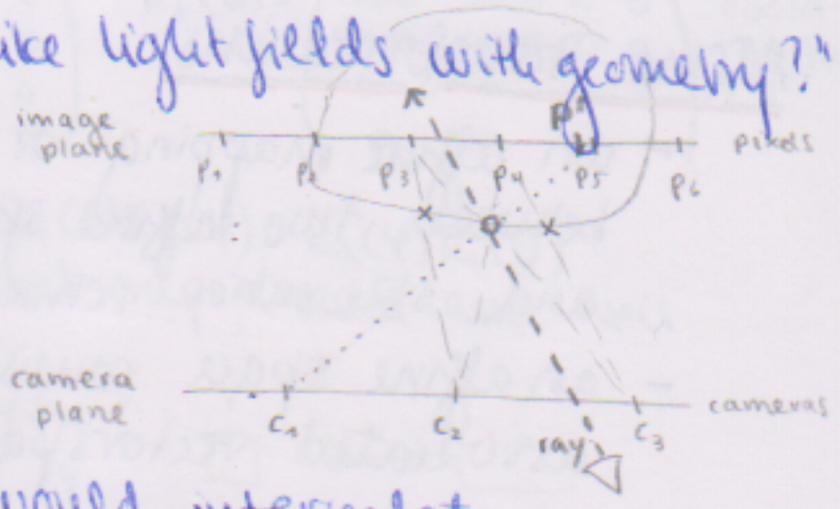
- we do the same iteration of image generation as in light fields but always change the light intensity/color



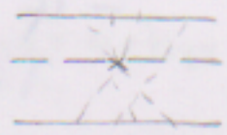
→ then we have many images of same scene where the lighting can be added for different outcomes
- could also be described as one image per incident light direction

Geometry-assisted IBR

"How can we improve image-based rendering like light fields with geometry?"



- we introduce depth-corrected rendering or lumigraph rendering
  - in addition to the light field we have a geometry model
  - normally (in standard light field) you would interpolate between c_2 and c_3 and p_3 and p_4 (quadrilinear interpolation) but we then generate wrong images
  - here you look up where the neighboring camerarays (c_2, c_3) intersect with the object, then find out at what point the ray hits the object surface (interpolation).
  - now look up ~~which~~ from your neighboring cameras (c_2, c_3) what ray goes through the ray-object-point (the rays don't need to land on pixels $p_1 \dots p_6$, they can also be interpolated) ⇒ interpolate between found rays
- synthetic aperture photography: with lumigraphs it is also possible to shift the aperture afterwards
  → generate depth-shifting images



- view-dependent texture mapping: map photos as textures onto 3D-models (interpolate between images from different directions)

- surface light fields (SLF): parameterize a light field over a 3D-model of an object. for any point on the surface and for every viewing direction, you can locate 3 nearby cameras and interpolate between the colors
  → if you have a whole dome of cameras, you have a data lumisphere, which can be faired