## The Story So Far

- The algorithms presented so far exploit:
  - Sparse sets of images (some data may not be available)
  - User help with correspondences (time consuming)
  - Extensive use of image warps
- The trade-off: Small amounts of data, more complex algorithms
- Sampling remains a problem
  - Images tend to appear blurry
  - Relatively little work on reconstruction algorithms

## Light Field Rendering or Lumigraphs

- Aims:
  - Sample the plenoptic function, or *light field*, densely
  - Store the samples in a data structure that is easy to access
  - Rendering is simply averaging of samples
- The plenoptic function gives the radiance passing through a point in space in a particular direction
- In free space: Gives the radiance along a *line*
  - Recall that radiance is constant along a line

## Storing Light Fields

- Each sample of the light field represents radiance along a line
- Required operations:
  - Store the radiance associated with an *oriented* line
  - Look up the radiance of lines that are “close” to a desired line
- Hence, we need some way of describing, or parameterizing, oriented lines
  - A line is a 4D object
  - There are several possible parameterizations

## Parameterizing Oriented Lines

- Desirable properties:
  - Efficient conversion from lines to parameters
  - Control over which subset of lines is of interest
  - Ease of uniform sampling of lines in space
- Parameterize lines by their intersection with two planes in arbitrary positions
  - Take *(s,t)* as intersection of line in one plane, *(u,v)* as intersection in other: *L(s,t,u,v)*
  - *Light Slab:* use two quadrilaterals (squares) and restrict each of *s,t,u,v* to (0,1)

## Line Space

- An alternate parameterization is line space
  - Better for looking at subset of lines and verifying sampling patterns
  - In 2D, parameterize lines by their angle with the x-axis, and their perpendicular distance form the origin
  - Extension to 3D is straightforward
- Every *line* in space maps to a *point* in line space, and vice versa
  - The two spaces are *dual*
  - Some operations are much easier in one space than the other

## Verifying Sampling Patterns

## Capturing Light Fields

- Render synthetic images
- Capture digitized photographs
  - Use a gantry to carefully control which images are captured
    - Makes it easy to control the light field sampling pattern
    - Hard to build the gantry
  - Use a video camera
    - Easy to acquire the images
    - Hard to control the sampling pattern

## Render synthetic images

- Decide which line you wish to sample, and cast a ray, or
- Render an array of images from points on the *(u,v)* plane – pixels in the images are points on the *(s,t)* plane
- Antialiasing is essential, both in *(s,t)* and *(u,v)*
  - Standard anitaliasing and aperture filtering

## Tightly Controlled Capture

- Use a computer controlled gantry to move a camera to fixed positions and take digital images
- Looks in at an object from outside
  - Must acquire multiple slabs to get full coverage
  - Care must be taken with camera alignment and optics
- Object is rotated in front of gantry to get multiple slabs
  - Must ensure lighting moves with the object
- Effectively samples light field on a regular grid, so rendering is easier

## Capture from Hand Held Video

- Place the object on a calibrated stage
  - Colored to allow blue-screening
  - Markers to allow easy determination of camera pose
- Wave the camera around in front of the object
  - Map to help guide where more samples are required
- Camera must be calibrated beforehand
- Output: A large number of non-uniform samples
- Problem: Have to re-sample to get regular sampling for rendering

## Re-Sampling the Light Field

- Basic problem:
  - Input: The set of irregular samples from the video capture process
  - Output: Estimates of the radiance on a regular grid in parameter space
- Algorithm outline:
  - Use a multi-resolution algorithm to estimate radiance in under-sampled regions
  - Use a binning algorithm to uniformly resample without bias

## Compression

- Light fields samples must be dense for good rendering
- Dense light fields are big: 1.6GB
  - When rendering, samples could come from any part of the light field
  - All of the light field must be in memory for real-time rendering
  - But lots of data redundancy, so compression should do well
- Desirable compression scheme properties:
  - Random access to compressed data
  - Asymmetric – slow compression, fast decompression

## Compression Scheme

- Vector Quantization:
  - Compression:
    - Choose a codebook of reproduction vectors
    - Replace all the vectors in the data with the index into the "nearest" vector in the codebook
  - Storage: The codebook plus the indexes
  - Decompression:
    - Replace each index with the vector from the codebook
- Follow up with Lempel-Ziv entropy encoding (gzip)
  - Decompress into memory

## Alternate Compression Schemes

- Neighboring "images" in (u,v) are likely to be very similar
  - Picture doesn't change much as you move the camera a little
  - We know what the camera motion is
  - BRDF changes smoothly for many cases
  - Use MPEG or similar to encode a sequence of images
  - This has been discussed but not implemented
- "Textures" should compress well
  - Use hardware rendering from compressed textures

## Rendering

- Ray-tracing: For each pixel in the image:
  - Determine the ray passing through the eye and the pixel
  - Interpolate the radiance along that ray from the nearest rays in the light-field
- Texture Mapping:
  - Finding the *(u,v)* and *(s,t)* coordinates is exactly the texture mapping operation
  - Use graphics hardware to do the job, or write a software texture mapper (maybe faster – only have to texture map two polygons)
- Use various interpolation schemes to control aliasing

## Exploiting Geometry

- When using the video capture approach, build a geometric model
  - Use a volume carving technique
- When determining the "nearest" samples for rendering, use the geometry to choose better samples
- This has been further extended:
  - Surface point used for improving sampling determines focus
  - By default, we want focus at the object, so use the object geometry
  - Using other surfaces gives depth of field and variable focus

## Surface Light Fields

- Instead of storing the complete light-field, store only lines emanating from the surface
  - Parameterize the surface mesh (standard technique)
  - Choose sample points on the surface
  - Sample the space of rays leaving the surface from those points
  - When rendering, look up nearby sample points and appropriate sample rays
- Best for rendering complex BRDF models
  - An example of *view dependent texturing*

## Summary

- Light-fields capture very dense representations of the plenoptic function
  - Fields can be stitched together to give walkthroughs
  - The data requirements are large
  - Sampling still not dense enough – filtering introduces blurring
- Next time: Using domain specific knowledge