

PBRT core

Digital Image Synthesis

Yung-Yu Chuang

9/28/2006

with slides by Pat Hanrahan

Announcements



- Please subscribe the mailing list.
- I will be attending Pacific Graphics 2006 from 10/11-10/13. The class of 10/12 will be cancelled.
- HW#1 will be assigned next week (10/5) and due on 10/26.

This course



- Study of how state-of-art ray tracers work



pbrt



- pbrt (physically-based ray tracing) attempts to simulate physical interaction between light and matter
- Plug-in architecture
- Core code performs the main flow and defines the interfaces to plug-ins. Necessary modules are loaded at run time as DLLs, so that it is easy to extend the system.

pbirt plug-ins



Table 1.1: Plug-ins. pbirt supports 13 types of plug-in objects that can be loaded at run time based on the contents of the scene description file. The system can be extended with new plug-ins, without needing to be recompiled itself.

Base class	Directory📁	Section
Shape	shapes/	3.1
Primitive	accelerators/	4.1
Camera	cameras/	6.1
Film	film/	8.1
Filter	filters/	7.6
Sampler	samplers/	7.2
ToneMap	tonemaps/	8.4
Material	materials/	10.2
Texture	textures/	11.3
VolumeRegion	volumes/	12.3
Light	lights/	13.1
SurfaceIntegrator	integrators/	16
VolumeIntegrator	integrators/	17

Example scene



```
LookAt 0 10 100 0 -1 0 0 1 0
Camera "perspective" "float fov" [30]
PixelFilter "mitchell"
    "float xwidth" [2] "float ywidth" [2]
Sampler "bestcandidate"
Film "image" "string filename" ["test.exr"]
    "integer xresolution" [200]
    "integer yresolution" [200] rendering options
# this is a meaningless comment
WorldBegin
id "type" param-list
    AttributeBegin
        CoordSysTransform "camera" "type name" [value]
        LightSource "distant"
            "point from" [0 0 0] "point to" [0 0 1]
            "color L" [3 3 3]
    AttributeEnd
```

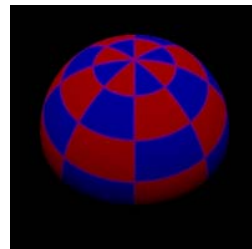
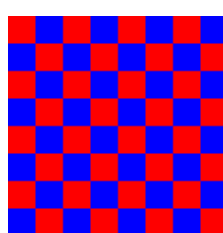
Example scene



```
AttributeBegin
  Rotate 135 1 0 0

  Texture "checks" "color" "checkerboard"
    "float uscale" [8] "float vscale" [8]
    "color tex1" [1 0 0] "color tex2" [0 0 1]

  Material "matte"
    "texture Kd" "checks"
  Shape "sphere" "float radius" [20]
AttributeEnd
WorldEnd
```



Phases of execution

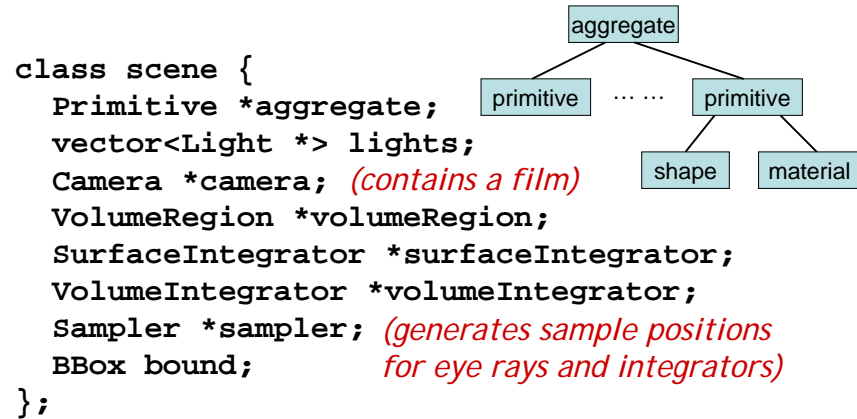


```
• main() in renderer/pbirt.cpp
int main(int argc, char *argv[]) {
  <Print welcome banner>
  pbirtInit();
  // Process scene description
  if (argc == 1) {
    // Parse scene from standard input
    ParseFile("-");
  } else {
    // Parse scene from input files
    for (int i = 1; i < argc; i++)
      if (!ParseFile(argv[i]))
        Error("Couldn't open ...\"%s\"\\n", argv[i]);
  }
  pbirtCleanup();
  return 0;
}
```

Scene parsing



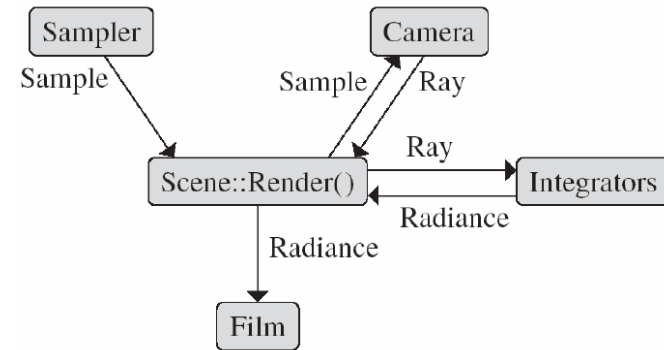
- core/pbrtlex.l and core/pbrtparse.y
- After parsing, a `scene` object is created (core/scene.*)



Rendering



- `Scene::Render()` is invoked.



Scene::Render()



```
while (sampler->GetNextSample(sample)) {
  RayDifferential ray;
  float rayW=camera->GenerateRay(*sample,&ray);

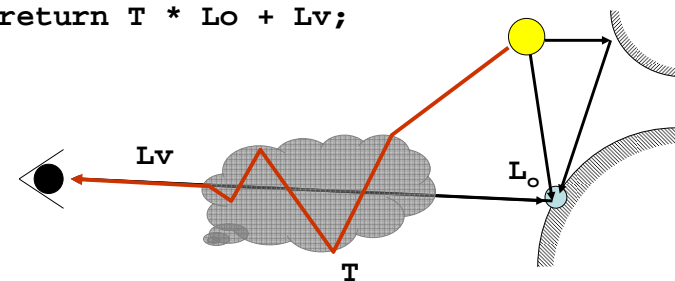
  <Generate ray differentials for camera ray>

  float alpha; opacity along the ray
  Spectrum Ls = 0.f;
  if (rayW > 0.f)
    Ls = rayW * Li(ray, sample, &alpha);
  ...
  camera->film->AddSample(*sample,ray,Ls,alpha);
  ...
}
```

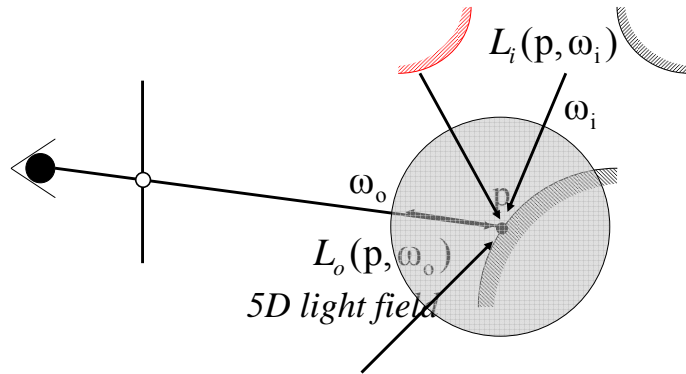
Scene::Li



```
Spectrum Scene::Li(RayDifferential &ray,
                  Sample *sample, float *alpha)
{
  Spectrum Lo=surfaceIntegrator->Li(...);
  Spectrum T=volumeIntegrator->Transmittance(...);
  Spectrum Lv=volumeIntegrator->Li(...);
  return T * Lo + Lv;
}
```

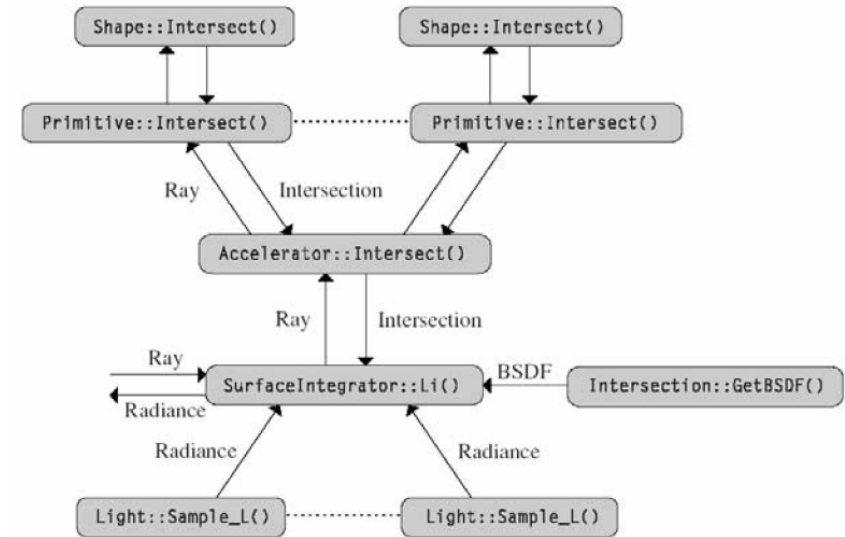


Rendering equation

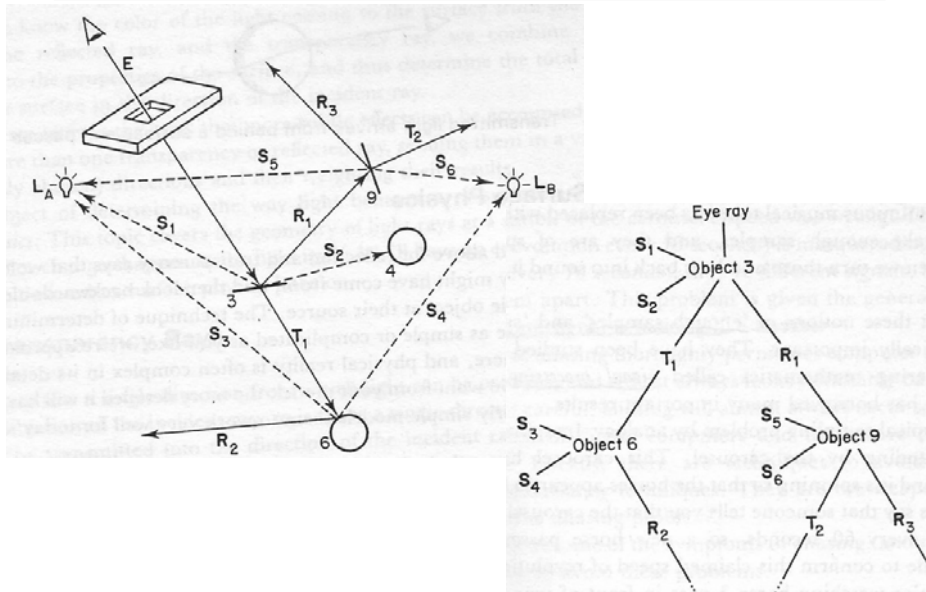


$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{s^2} f(p, \omega_o, \omega_i) L_i(p, \omega_i) |\cos \theta_i| d\omega_i$$

Surface integrator



Whitted model



Whitted integrator



- in integrators/whitted.cpp

```

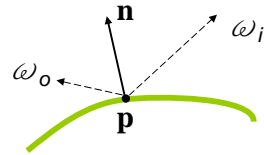
class WhittedIntegrator:public SurfaceIntegrator

Spectrum WhittedIntegrator::Li(Scene *scene,
    RayDifferential &ray, Sample *sample, float *alpha)
{
    ...
    bool hitSomething=scene->Intersect(ray,&isect);
    if (!hitSomething) {include effects of light without geometry}
    else {
        ...
        <Computed emitted and reflect light at isect>
    }
}
    
```

Whitted integrator



```
BSDF *bsdf=isect.GetBSDF(ray);
...
Vector wo=-ray.d;
L+=isect.Le(wo);
Vector wi; direct lighting
for (u_int i = 0; i < scene->lights.size(); ++i) {
    VisibilityTester visibility;
    Spectrum Li = scene->lights[i]->
        Sample_L(p, &wi, &visibility);
    if (Li.Black()) continue;
    Spectrum f = bsdf->f(wo, wi);
    if (!f.Black() && visibility.Unoccluded(scene))
        L += f * Li * AbsDot(wi, n) *
            visibility.Transmittance(scene);
}
```



Whitted integrator



```
if (rayDepth++ < maxDepth) {
    Spectrum f = bsdf->Sample_f(wo, &wi,
        BxDFType(BSDF_REFLECTION | BSDF_SPECULAR));
    if (!f.Black()) {
        <compute rd for specular reflection>
        L += scene->Li(rd, sample) * f * AbsDot(wi, n);
    }
    f = bsdf->Sample_f(wo, &wi,
        BxDFType(BSDF_TRANSMISSION | BSDF_SPECULAR));
    if (!f.Black()) {
        <compute rd for specular transmission>
        L += scene->Li(rd, sample) * f * AbsDot(wi, n);
    }
}
```

Code optimization



- Two commonly used tips
 - Divide, square root and trigonometric are among the slowest (10-50 times slower than +*). Multiplying 1/r for dividing r.
 - Being cache conscious

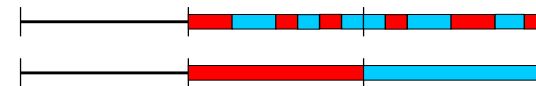
Cache-conscious programming



- `alloca`
- `AllocAligned()`, `FreeAligned()` make sure that memory is cache-aligned



- Use union and bitfields to reduce size and increase locality
- Split data into hot and cold



Cache-conscious programming



- Arena-based allocation allows faster allocation and better locality because of contiguous addresses.
- Blocked 2D array, used for film

