

Materials

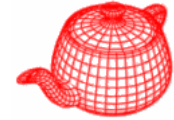
Digital Image Synthesis

Yung-Yu Chuang

11/16/2006

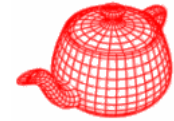
with slides by Robin Chen

Materials



- Reflection models in the previous chapter don't describe how light is scattered at a particular point and their parameters.
- A surface shader, represented by **Material**, is bound to each primitive in the scene.
- **Material=BSDF+Texture** (canned materials)
- **core/material.* materials/***

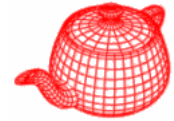
BSDFs



- **BSDF**=a collection of **BxDF** (BRDFs and BTDFs)
- A real material is likely a mixture of several specular, diffuse and glossy components (pbrt sets 8 as the maximum number of components)

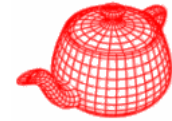
```
class BSDF {  
    ...  
private:  
    Normal nn, ng; // shading normal, geometry normal  
    Vector sn, tn; // shading tangents  
    int nBxDFs;  
    #define MAX_BxDFS 8  
    BxDF * bxdfs[MAX_BxDFS];  
    static MemoryArena arena;  
};
```

BSDF



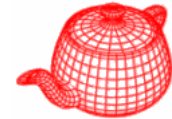
```
BSDF::BSDF(const DifferentialGeometry &dg,  
           const Normal &ngeom, float e)  
  : dgShading(dg), eta(e) {  
  ng = ngeom;  
  nn = dgShading.nn;  
  sn = Normalize(dgShading.dpdu);  
  tn = Cross(nn, sn);  
  nBxDFs = 0;  
}  
  
inline void BSDF::Add(BxDF *b) {  
  Assert(nBxDFs < MAX_BxDFs);  
  bxdfs[nBxDFs++] = b;  
}
```

BSDF



```
Spectrum BSDF::f(const Vector &woW, const Vector &wiW,
                BxDFType flags) {
    Vector wi=WorldToLocal(wiW), wo=WorldToLocal(woW);
    if (Dot(wiW, ng) * Dot(woW, ng) > 0)
        // ignore BTDFs
        flags = BxDFType(flags & ~BSDF_TRANSMISSION);
    else
        // ignore BRDFs
        flags = BxDFType(flags & ~BSDF_REFLECTION);
    Spectrum f = 0.;
    for (int i = 0; i < nBxDFs; ++i)
        if (bxdfs[i]->MatchesFlags(flags))
            f += bxdfs[i]->f(wo, wi);
    return f;
}
```

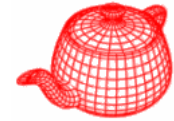
Material



- `Material::GetBSDF()` determines the reflective properties for a given point on the surface.

```
class Material : public ReferenceCounted {
public:
    real geometry around intersection
    virtual BSDF *GetBSDF(DifferentialGeometry &dgGeom,
        DifferentialGeometry &dgShading) const = 0;
    virtual ~Material();    shading geometry around intersection
    static void Bump(Reference<Texture<float> > d,
        const DifferentialGeometry &dgGeom,
        const DifferentialGeometry &dgShading,
        DifferentialGeometry *dgBump);
};
```

Matte

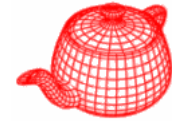


- Purely diffuse surface

```
class Matte : public Material {
public:
    Matte(Reference<Texture<Spectrum> > kd,
          Reference<Texture<float> > sig,
          Reference<Texture<float> > bump)
    { Kd = kd; sigma = sig; bumpMap = bump; }
    BSDF *GetBSDF(const DifferentialGeometry &dgGeom,
                  const DifferentialGeometry &dgShading) const;
private:
    Reference<Texture<Spectrum> > Kd;
    Reference<Texture<float> > sigma, bumpMap;
};
```

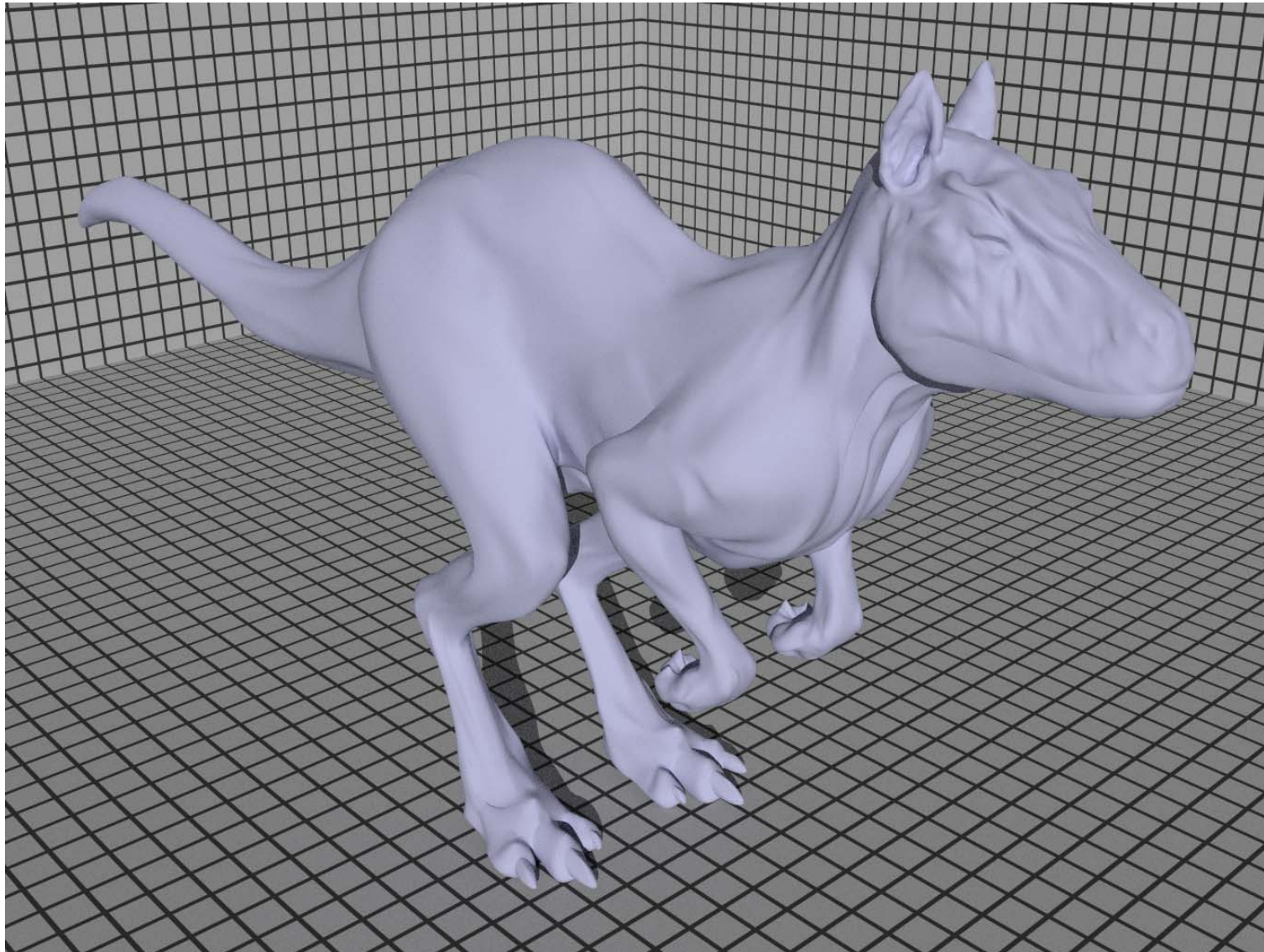
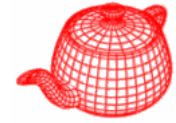
essentially a height map

Matte

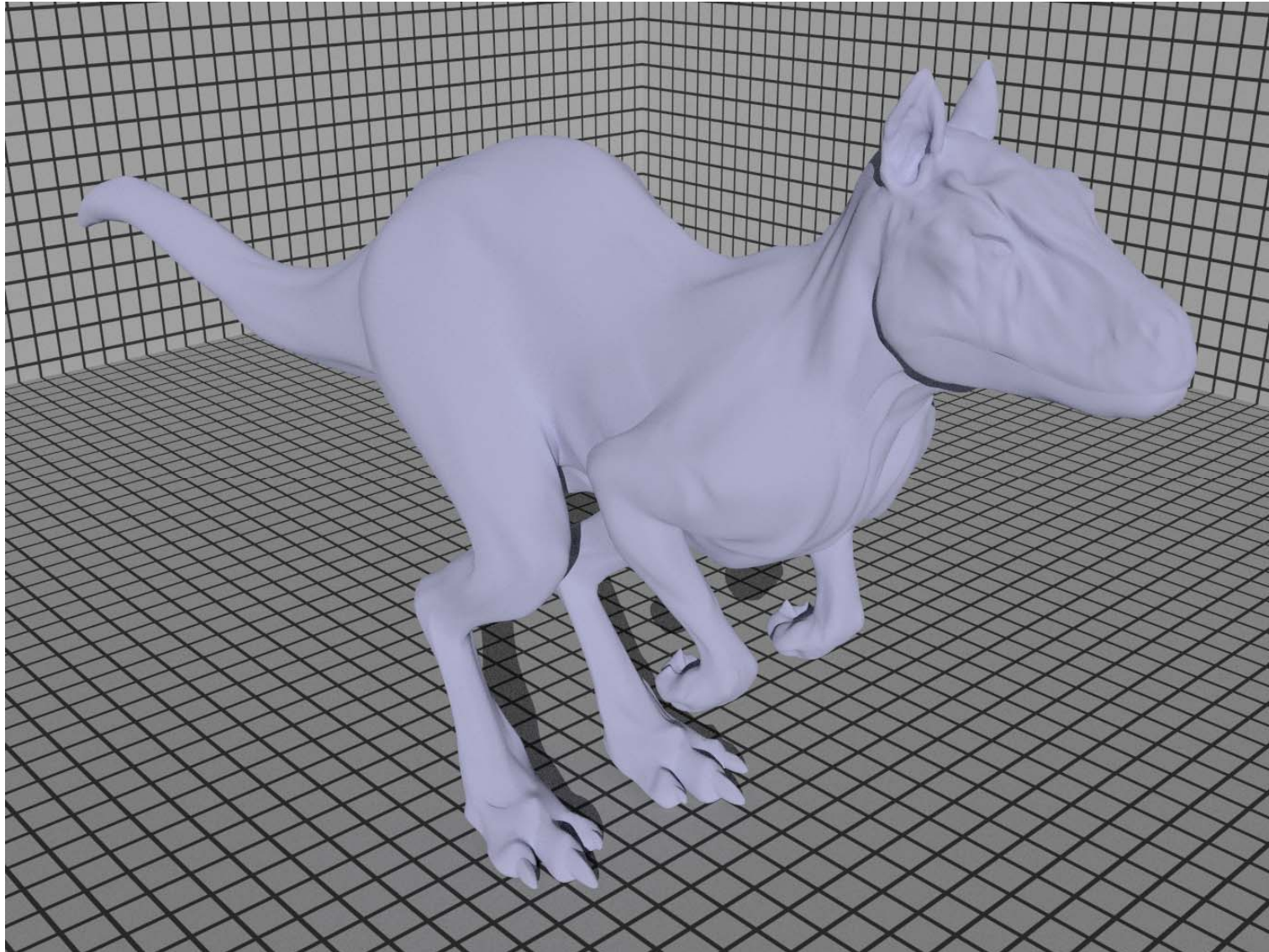
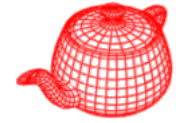


```
BSDF *Matte::GetBSDF(DifferentialGeometry &dgGeom,  
                    DifferentialGeometry &dgShading) {  
    DifferentialGeometry dgs;  
    if (bumpMap) Bump(bumpMap, dgGeom, dgShading, &dgs);  
    else dgs = dgShading;  
    BSDF *bsdf = BSDF_ALLOC(BSDF)(dgs, dgGeom.nn);  
  
    Spectrum r = Kd->Evaluate(dgs).Clamp();  
    float sig = Clamp(sigma->Evaluate(dgs), 0.f, 90.f);  
    if (sig == 0.)  
        bsdf->Add(BSDF_ALLOC(Lambertian)(r));  
    else  
        bsdf->Add(BSDF_ALLOC(OrenNayar)(r, sig));  
    return bsdf;  
}
```

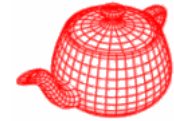
Lambertian



Oren-Nayer model



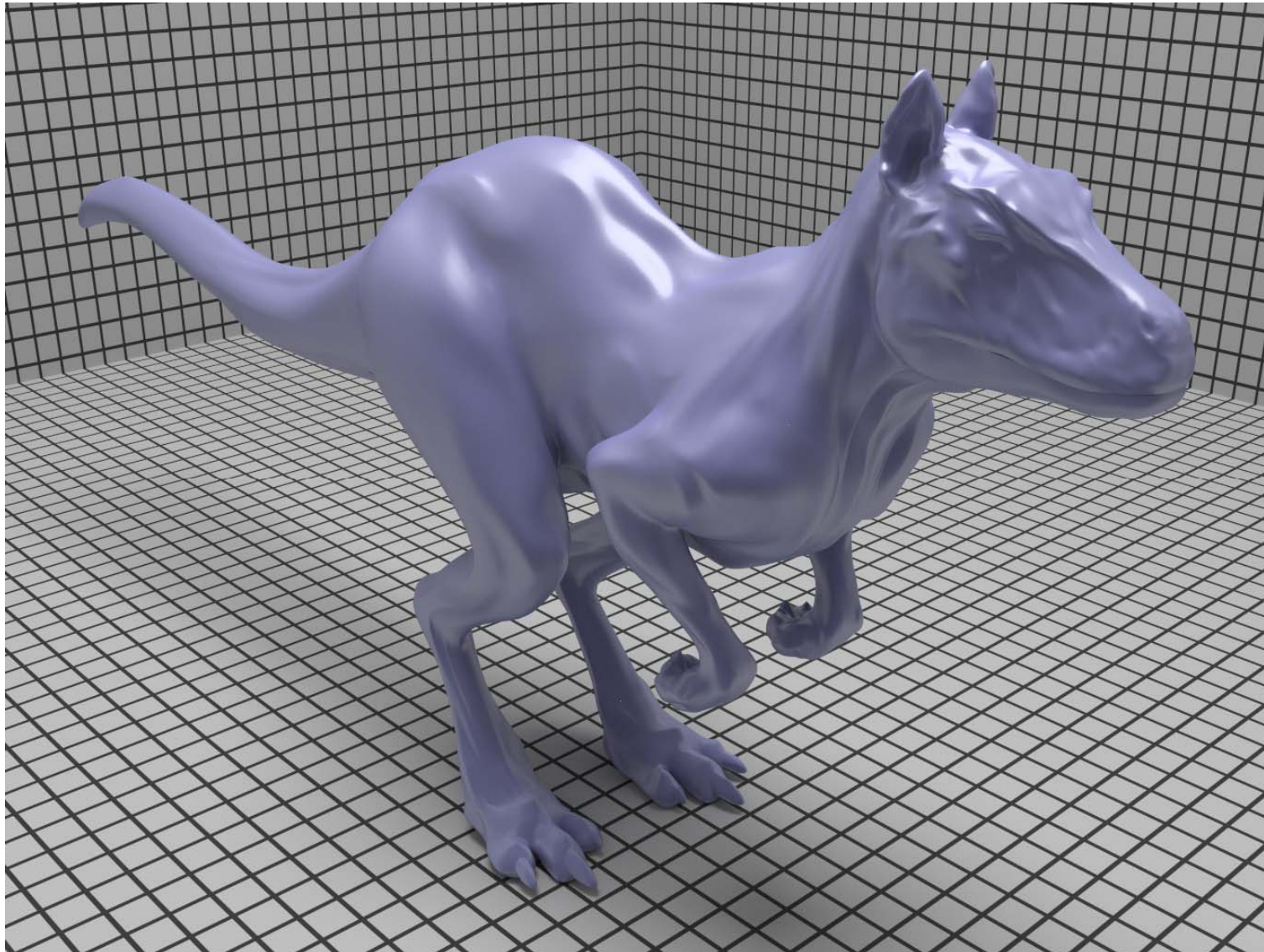
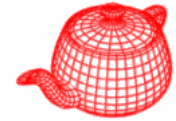
Plastic



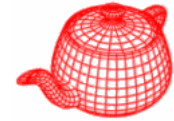
- A mixture of a diffuse and glossy scattering function with parameters, **kd**, **ks** and **rough**

```
BSDF *Plastic::GetBSDF(...) {  
    <Allocate BSDF, possibly doing bump-mapping>  
    Spectrum kd = Kd->Evaluate(dgs).Clamp();  
    BxDF *diff = BSDF_ALLOC(Lambertian)(kd);  
  
    Fresnel *f=BSDF_ALLOC(FresnelDielectric)(1.5f,1.f);  
    Spectrum ks = Ks->Evaluate(dgs).Clamp();  
    float rough = roughness->Evaluate(dgs);  
    BxDF *spec = BSDF_ALLOC(Microfacet)(ks, f,  
                                     BSDF_ALLOC(Blinn)(1.f / rough));  
  
    bsdf->Add(diff); bsdf->Add(spec); return bsdf;  
}
```

Plastic

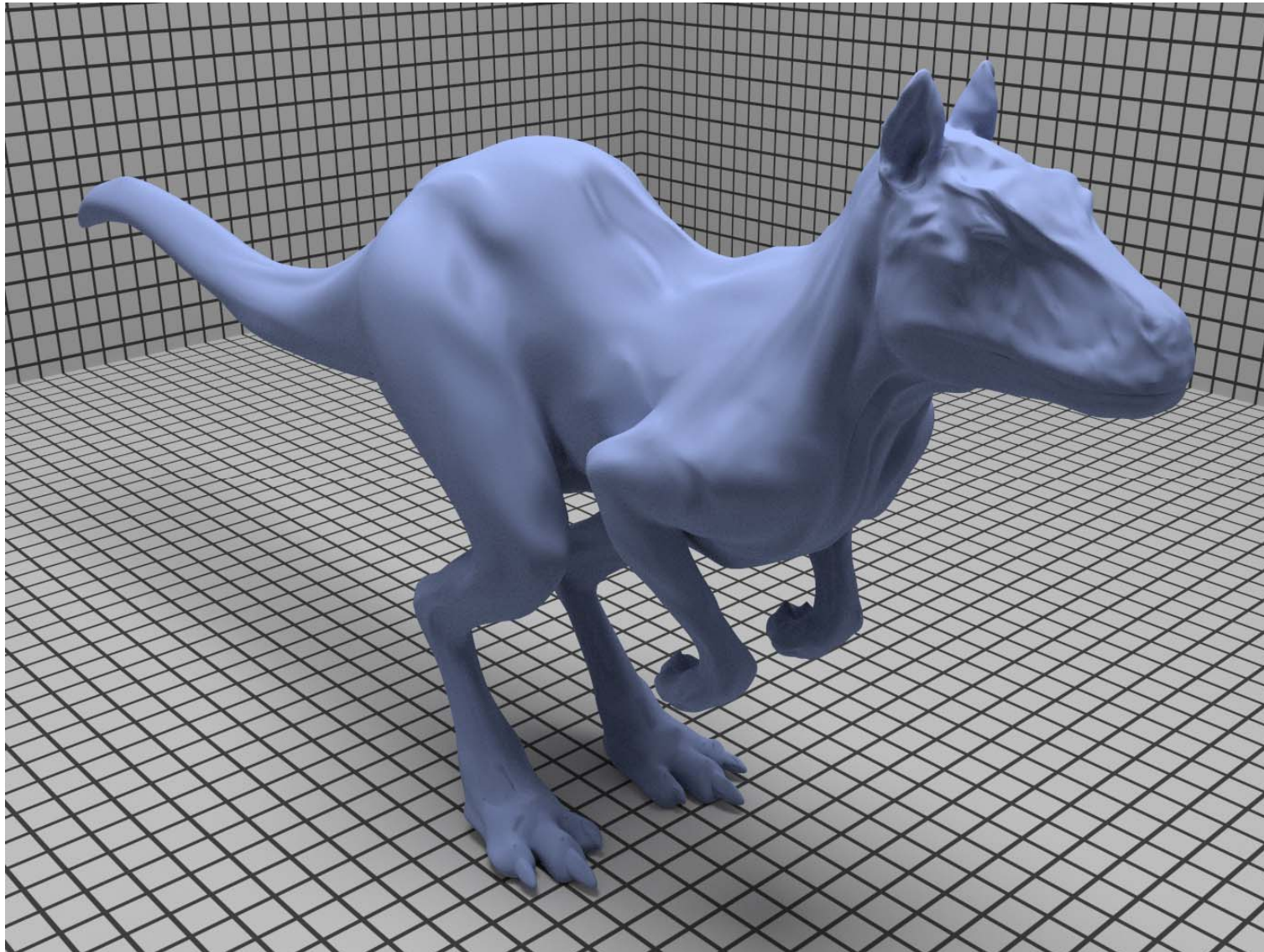
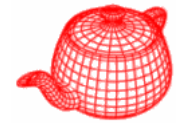


Additional materials

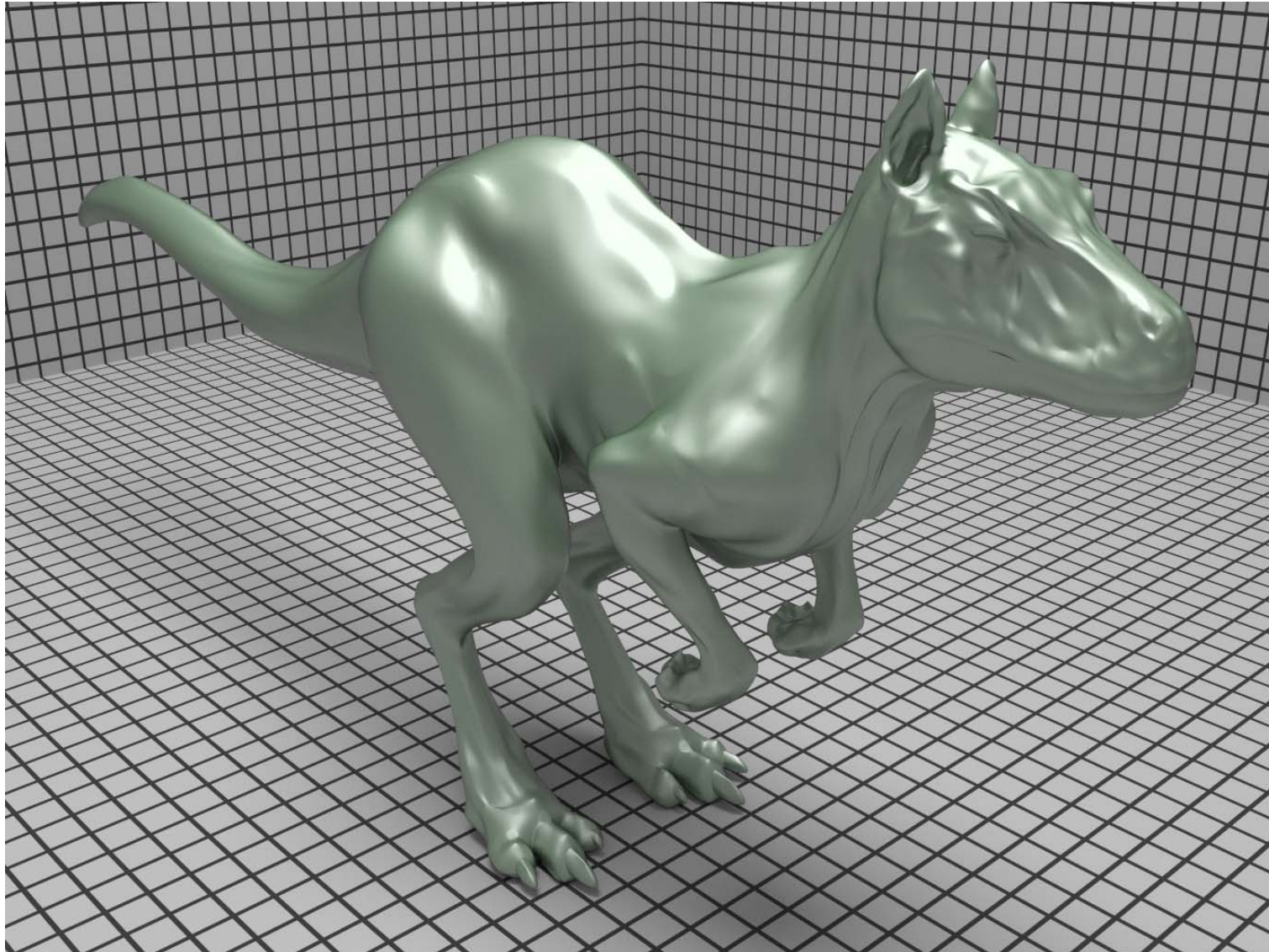
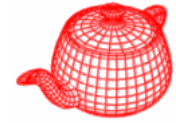


- There are totally 14 material plug-ins available in pbrt. Most of them are just variations of **Matte** and **Plastic**.
- **Translucent**: glossy transmission
- **Mirror**: perfect specular reflection
- **Glass**: reflection and transmission, Fresnel weighted
- **ShinyMetal**:
- **Substrate**: layered-model
- **Clay, Felt, Primer, Skin, BluePaint, Brushed Metal**: measured data fitted by Lafortune
- **Uber**: a “union” of previous material; highly parameterized

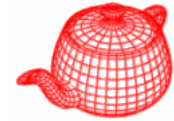
Blue paint (measured Lafortune)



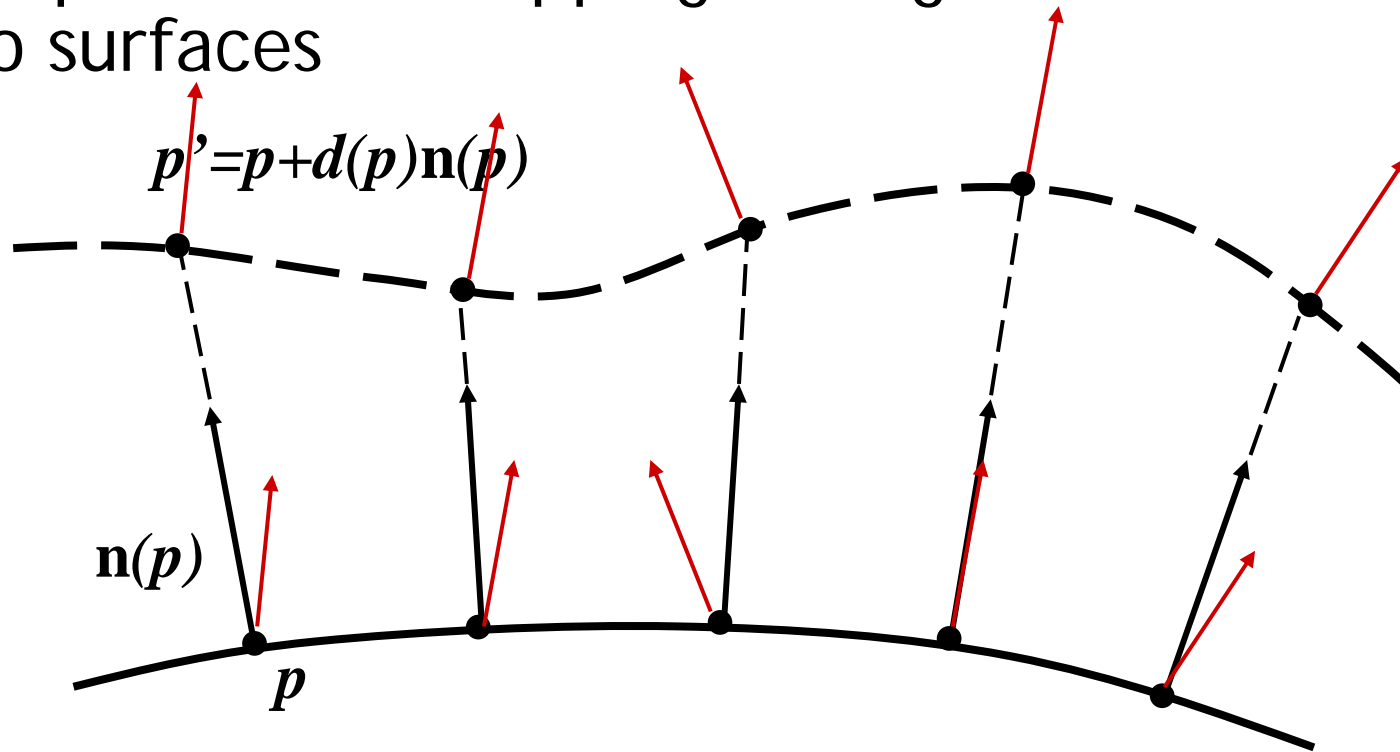
Substrate



Bump mapping

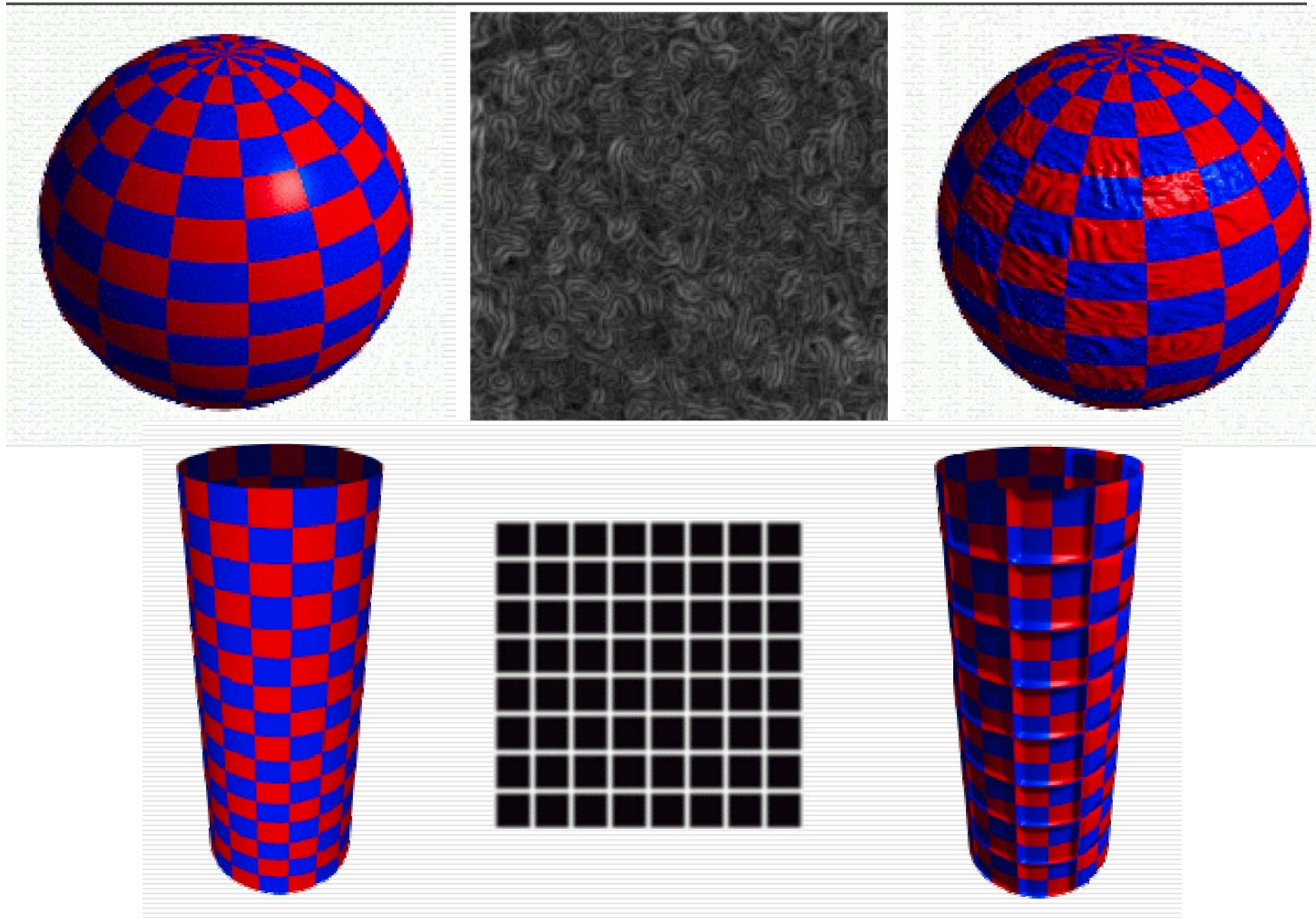
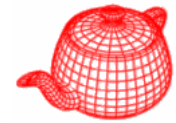


- Displacement mapping adds geometrical details to surfaces

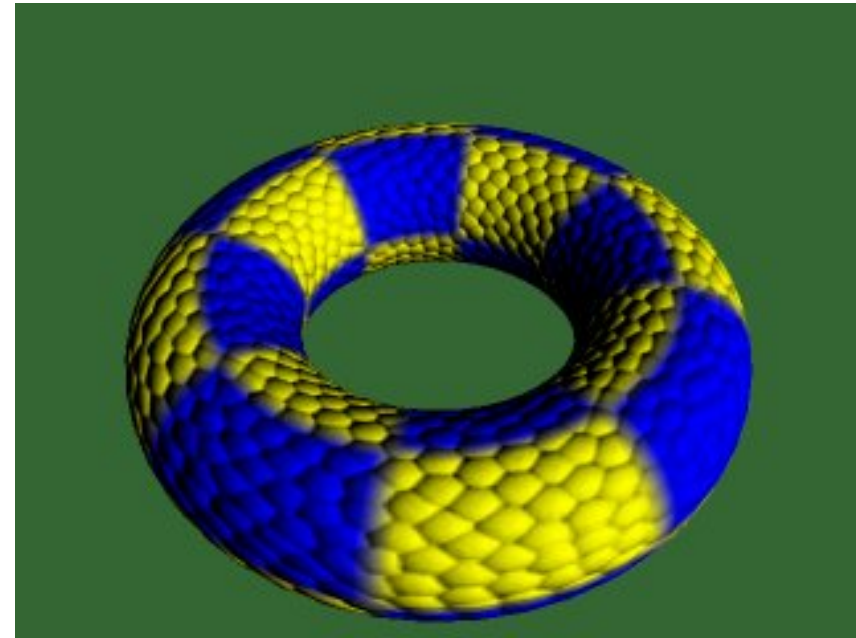
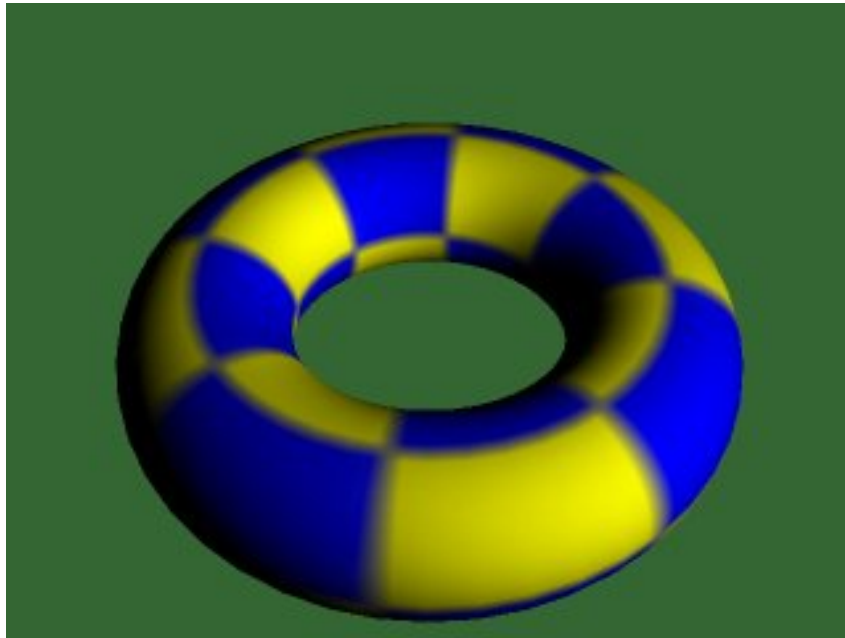
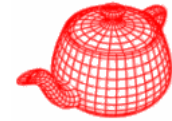


- Bump mapping: augments geometrical details to a surface without changing the surface itself
- It works well when the displacement is small

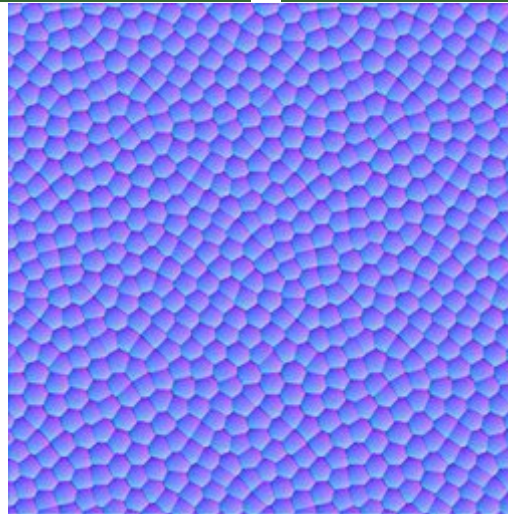
Bump mapping



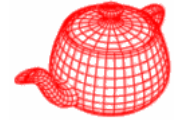
Bump mapping



To use bump mapping,
a displacement map is
often converted into a
bump map (normal map)



Bump mapping



$$\mathbf{q}(u, v) = \mathbf{p}(u, v) + d(u, v)\mathbf{n}(u, v)$$

$$\mathbf{n}' = \frac{\partial \mathbf{q}}{\partial u} \times \frac{\partial \mathbf{q}}{\partial v}$$

the only unknown term

$$\frac{\partial \mathbf{q}}{\partial u} = \frac{\partial \mathbf{p}(u, v)}{\partial u} + \frac{\partial d(u, v)}{\partial u} \mathbf{n}(u, v) + d(u, v) \frac{\partial \mathbf{n}(u, v)}{\partial u}$$

$$\frac{\partial \mathbf{q}}{\partial u} \approx \frac{\partial \mathbf{p}}{\partial u} + \frac{d(u + \Delta_u, v) - d(u, v)}{\Delta_u} \mathbf{n} + d(u, v) \frac{\partial \mathbf{n}}{\partial u}$$

often ignored

Material::Bump(...) does the above calculation



without bump mapping



with bump mapping