CSE 167: Introduction to Computer Graphics Lecture #19: Wrapping Up

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2017

Announcements

- TA evaluations
- CAPE
- Final project blog entries due:
  - Monday, Dec 4th at 11:59pm
  - Monday, Dec 12th at 11:59pm
- Video due:
  - Wednesday, Dec 13th at 12 noon

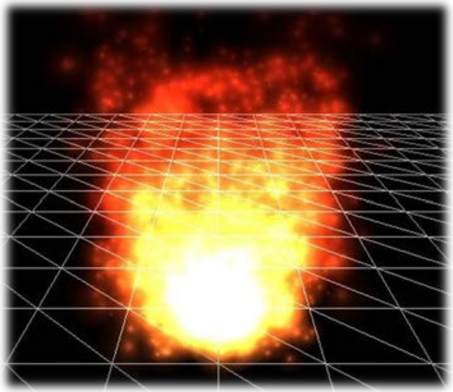
Lecture Overview

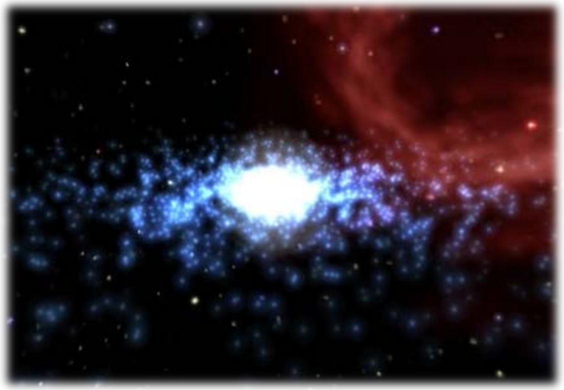
- Particle Systems
- Collision Detection
- Bump Mapping

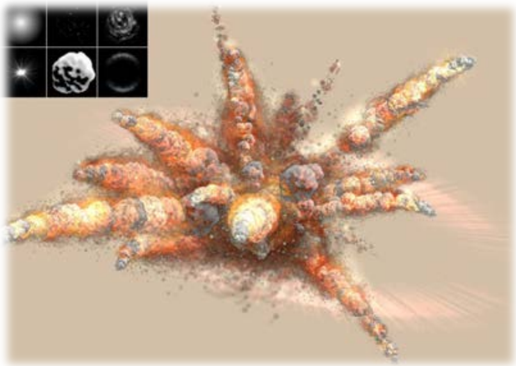
Particle Systems

Particle Systems

- Used for:
  - Fire/sparks
  - Rain/snow
  - Water spray
  - Explosions
  - Galaxies







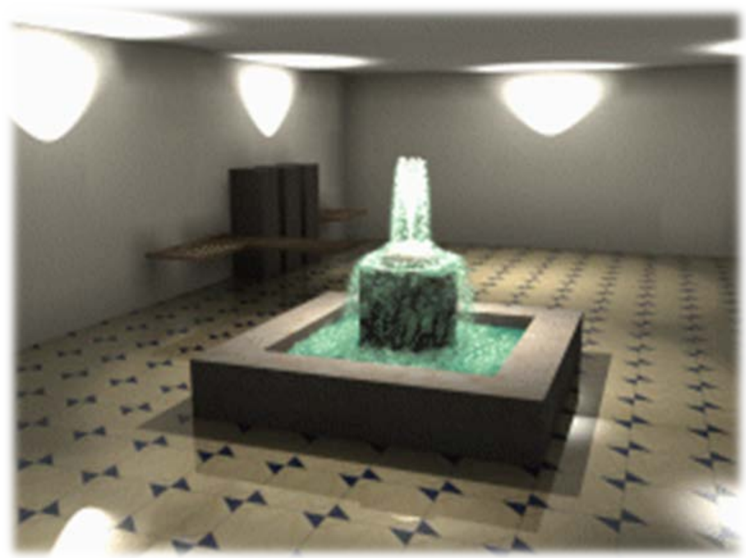
UCSD

Internal Representation

- Particle system is collection of a number of individual elements (particles)
  - Controls a set of particles which act autonomously but share some common attributes
- Particle Emitter: Source of all new particles
  - 3D point
  - Polygon mesh: particles' initial velocity vector is normal to surface
- Particle attributes:
  - position (3D)
  - velocity (vector: speed and direction)
  - color + opacity
  - lifetime
  - size
  - shape
  - weight

Dynamic Updates

- Particles change position and/or attributes with time
- Initial particle attributes often created with random numbers
- Frame update:
  - Parameters: simulation of particles, can include collisions with geometry
    - Forces (gravity, wind, etc) accelerate a particle
    - Acceleration changes velocity
    - Velocity changes position
  - Rendering:
    - GL_POINTS
    - GL_POINT_SPRITE
    - Point shader



Source: http://www.particlesystems.org/

Point Rendering – Vertex Shader

```
uniform mat4 u_MVPMatrix;
uniform vec3 u_cameraPos;

// Constants (tweakable):
const float minPointScale = 0.1;
const float maxPointScale = 0.7;
const float maxDistance   = 100.0;

void main()
{
    // Calculate point scale based on distance from the viewer
    // to compensate for the fact that gl_PointSize is the point
    // size in rasterized points / pixels.
    float cameraDist = distance(a_position_size.xyz, u_cameraPos);
    float pointScale = 1.0 - (cameraDist / maxDistance);
    pointScale = max(pointScale, minPointScale);
    pointScale = min(pointScale, maxPointScale);

    // Set GL globals and forward the color:
    gl_Position  = u_MVPMatrix * vec4(a_position_size.xyz, 1.0);
    gl_PointSize = a_position_size.w * pointScale;
    v_color      = a_color;
}
```

Demo

- Particle system in WebGL:
  - http://nullprogram.com/webgl-particles/

UCSD

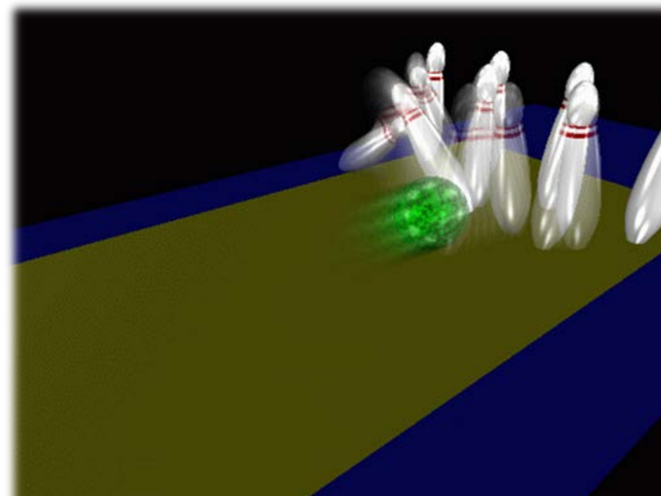
References

- Tutorial with source code by Bartlomiej Filipek, 2014:
  - http://www.codeproject.com/Articles/795065/Flexible-particle-system-OpenGL-Renderer
- Articles with source code:
  - Jeff Lander: "The Ocean Spray in Your Face", Game Developer, July 1998
    - http://www.darwin3d.com/gamedev/articles/col0798.pdf
  - John Van Der Burg: "Building an Advanced Particle System", Gamasutra, June 2000
    - http://www.gamasutra.com/view/feature/3157/building_an_advanced_particle_.php
- Founding scientific paper:
  - Reeves: "Particle Systems - A Technique for Modeling a Class of Fuzzy Objects", ACM Transactions on Graphics (TOG) Volume 2 Issue 2, April 1983
    - https://www.evl.uic.edu/aej/527/papers/Reeves1983.pdf

Collison Detection

Collision Detection

- Goals:
  - Physically correct simulation of collision of objects
    - Not covered here
  - Determine if two objects intersect
- Slow calculation because of exponential growth $O(n^2)$:
  - # collision tests = n*(n-1)/2



Intersection Testing

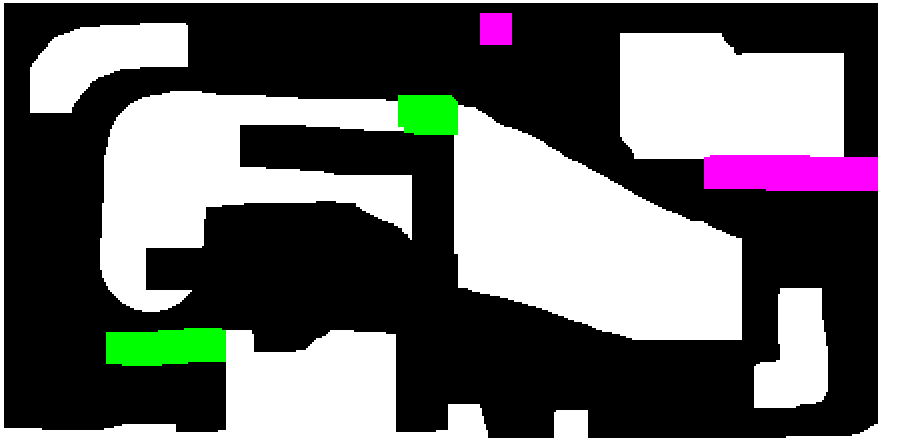
- Purpose:
  - Keep moving objects on the ground
  - Keep moving objects from going through walls, each other, etc.
- Goal:
  - Believable system, does not have to be physically correct
- Priority:
  - Computationally inexpensive
- Typical approach:
  - Spatial partitioning
  - Object simplified for collision detection by one or a few
    - Points
    - Spheres
    - Axis aligned bounding box (AABB)
  - Pairwise checks between points/spheres/AABBs and static geometry

Sweep and Prune Algorithm

- Sorts bounding boxes
- Not intuitively obvious how to sort bounding boxes in 3-space
- Dimension reduction approach:
  - Project each 3-dimensional bounding box onto the x,y and z axes
  - Find overlaps in 1D: a pair of bounding boxes can overlap if and only if their intervals overlap in all three dimensions
    - Construct 3 lists, one for each dimension
    - Each list contains start/end point of intervals corresponding to that dimension
    - By sorting these lists, we can determine which intervals overlap
    - Reduce sorting time by keeping sorted lists from previous frame, changing only the interval endpoints

Collision Map (CM)

- 2D map with information about where objects can go and what happens when they go there
- Colors indicate different types of locations
- Map can be computed from 3D model, or hand drawn with paint program
- Granularity: defines how much area (in object space) one CM pixel represents



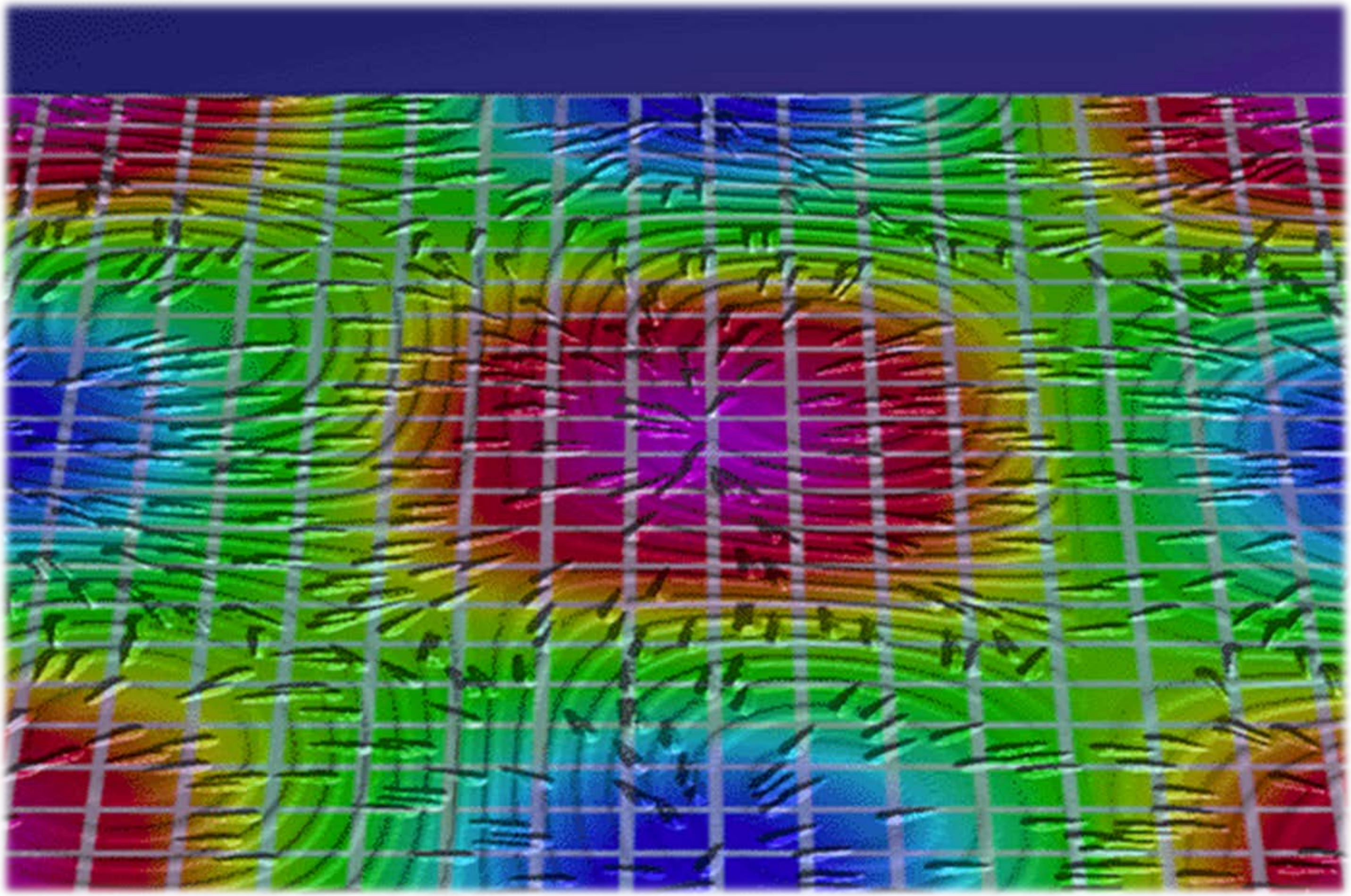
Bump Mapping

Bump Mapping

- Many textures are the result of small perturbations in the surface geometry
- Modeling these changes would result in an explosion in the number of geometric primitives.
- Bump mapping attempts to alter the lighting across a polygon to provide the illusion of texture.

[This chapter includes slides by Roger Crawfis]

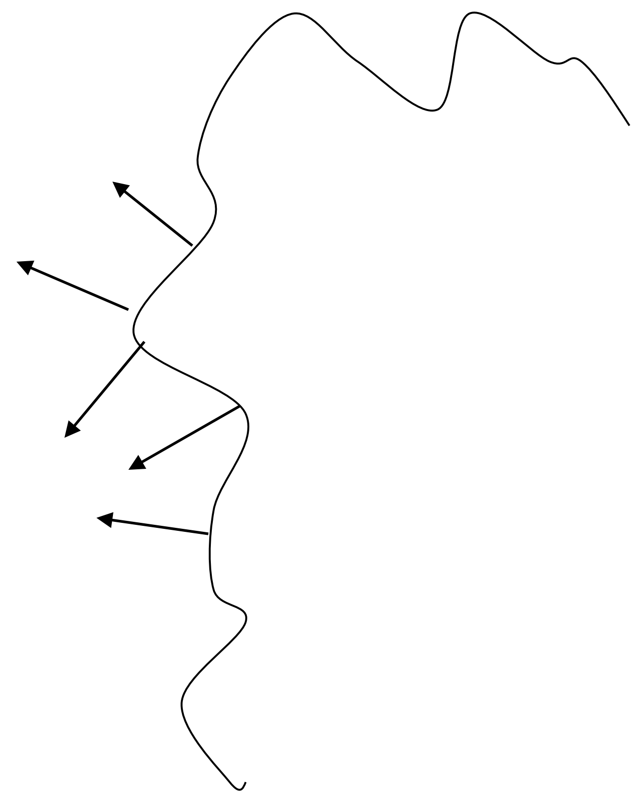
Bump Mapping Example



Crawfis 1991

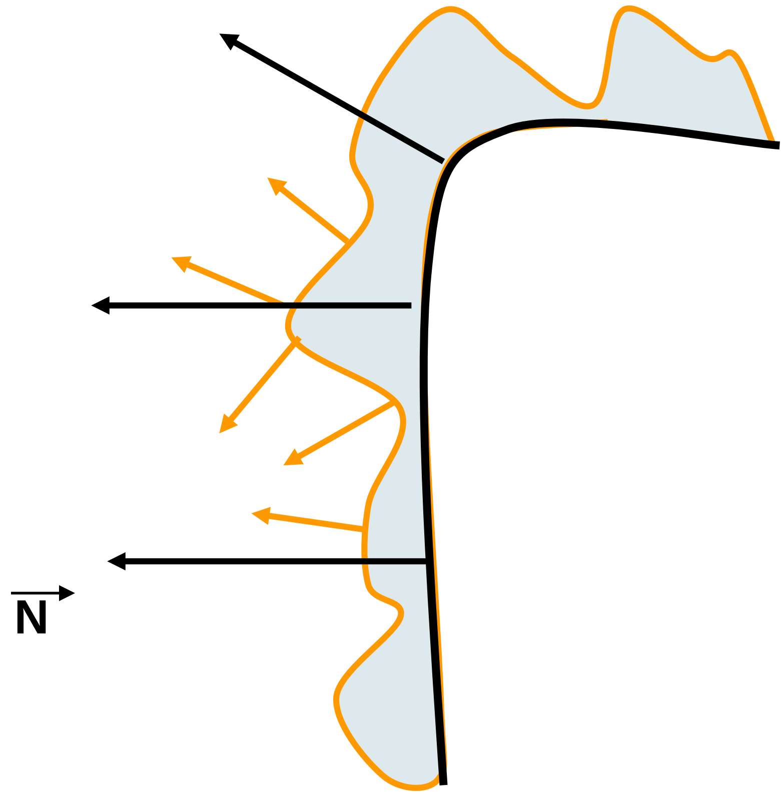
Bump Mapping

- Consider the lighting for a modeled surface.



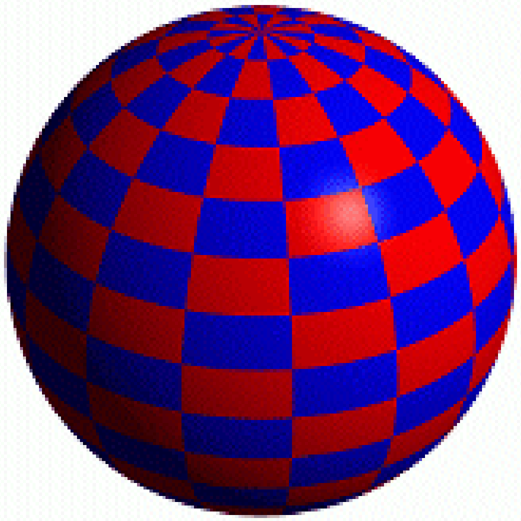
Bump Mapping

- We can model this as deviations from some base surface.
- The question is then how these deviations change the lighting.

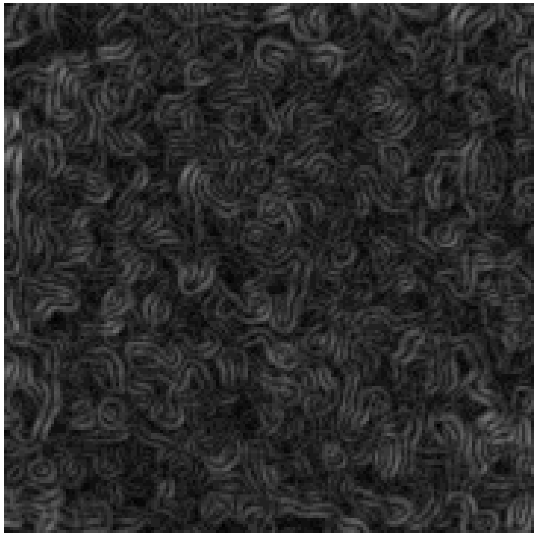


Bump Mapping

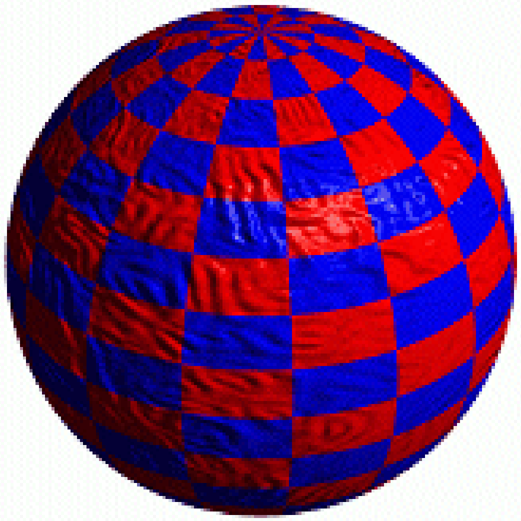
- Store in a texture and use textures to alter the surface normal
  - Does not change the shape of the surface
  - Just shaded as if it were a different shape



Sphere w/Diffuse Texture

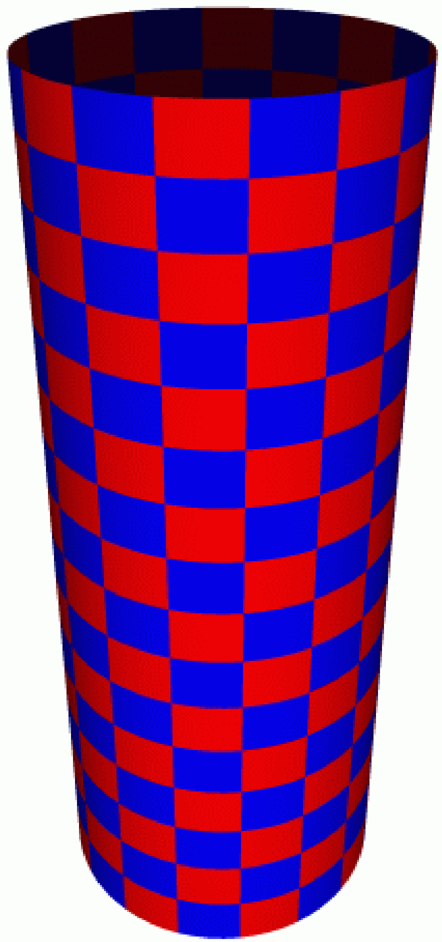


Swirly Bump Map

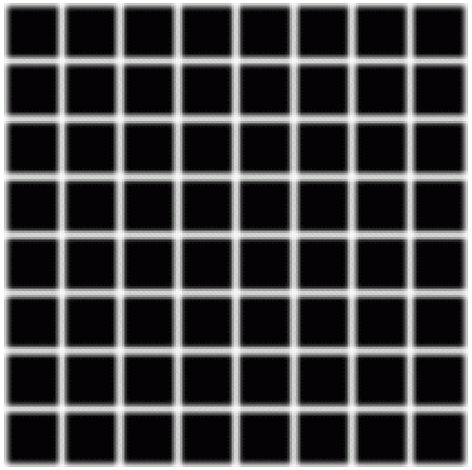


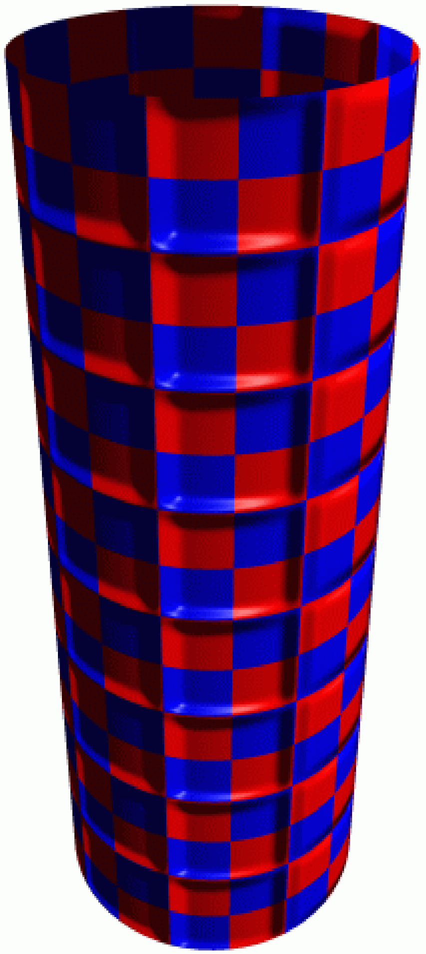
Sphere w/Diffuse Texture & Bump Map

Simple textures work great



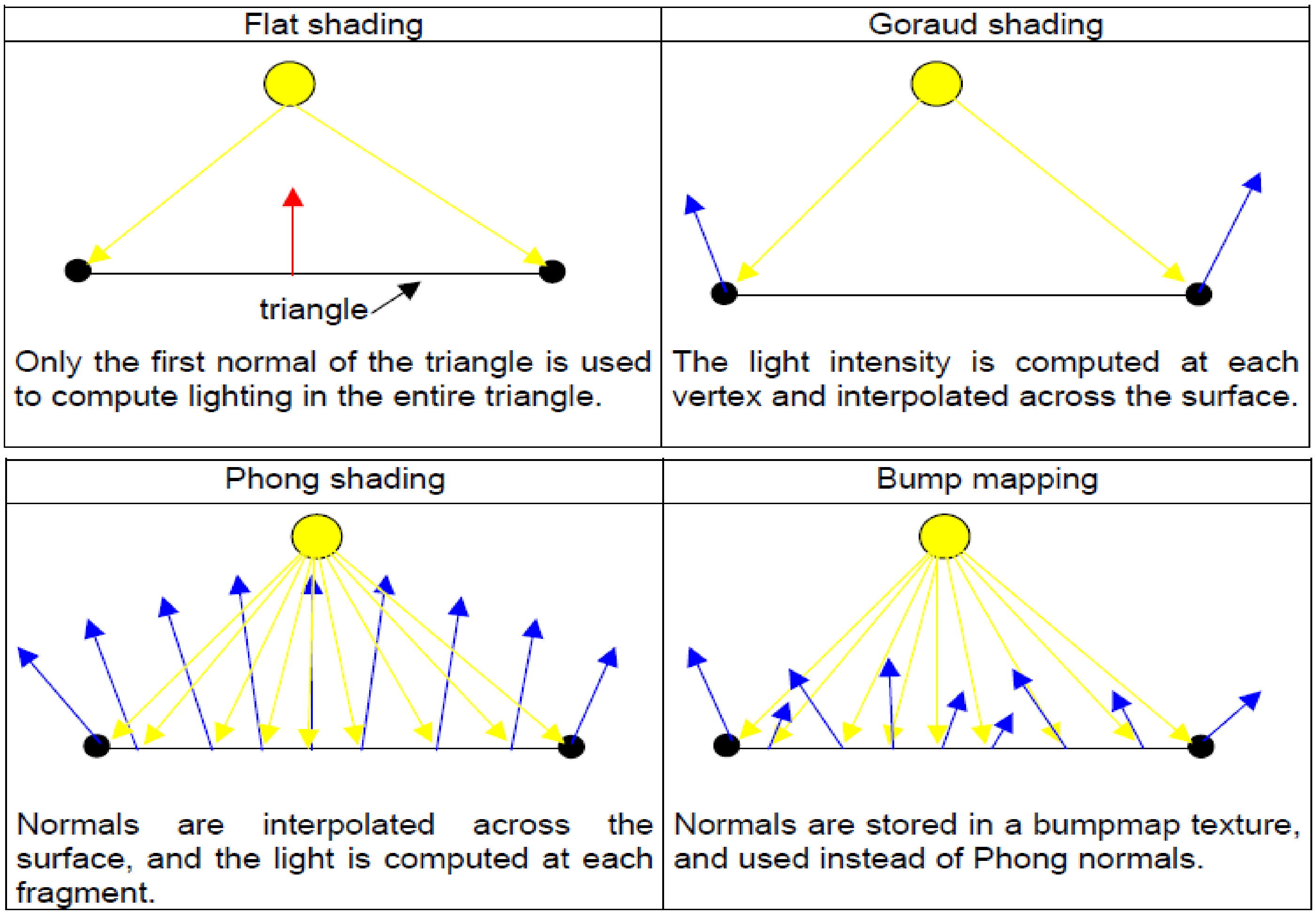
Cylinder w/Diffuse Texture Map





Cylinder w/Texture Map & Bump Map

Normal Mapping



Normal Mapping

Just texture mapped

Texture and normal maps

Notice: The geometry is unchanged. There's the same number of vertices and triangles. This effect is entirely from the normal map.

Normal Maps

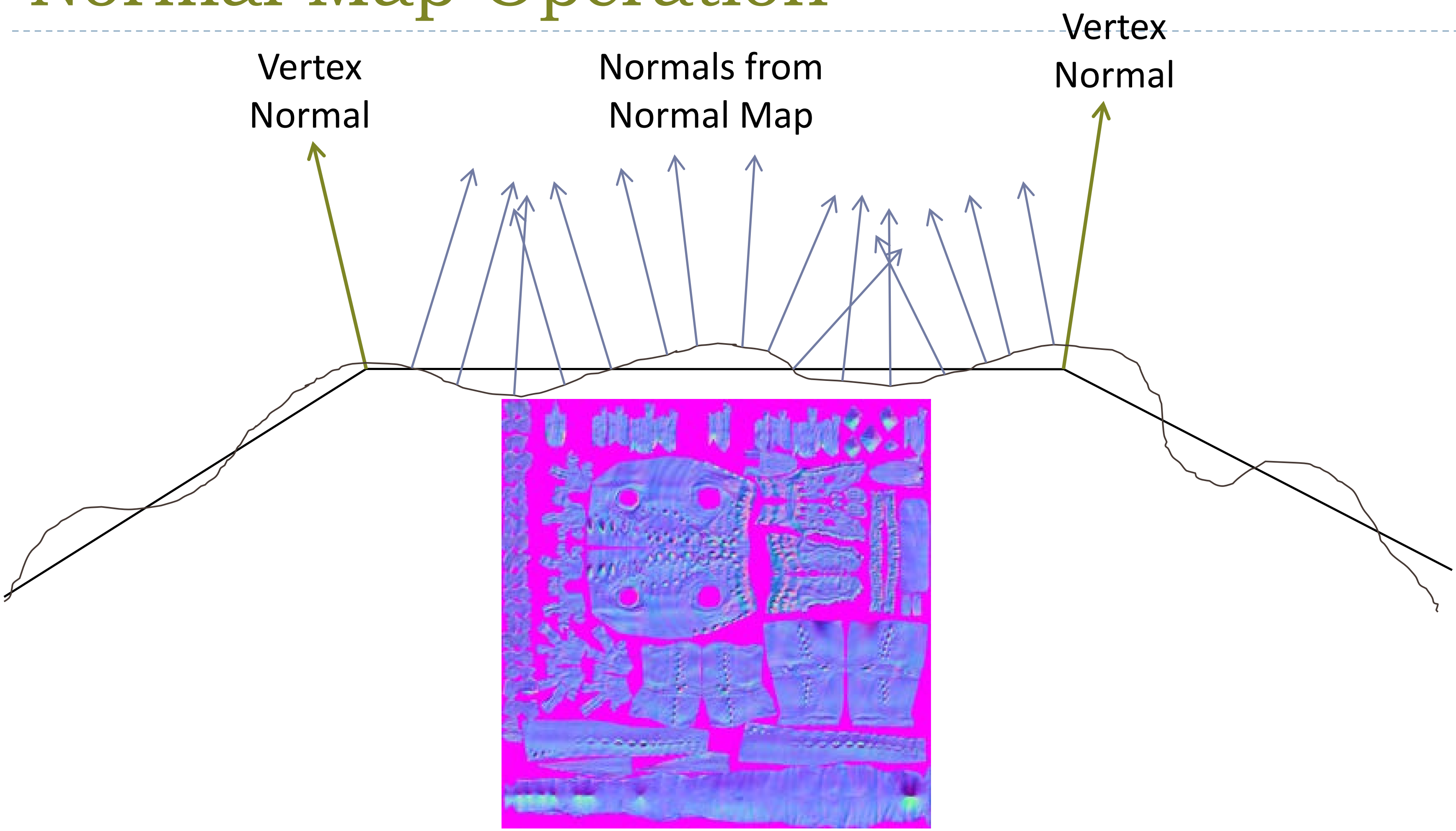
Diffuse Color Texture Map

Normal Map

Each pixel represents a normal vector relative to the surface at that point. -1 to 1 range is mapped to 0 to 1 for the texture so normals become colors.

→ Inverse of Normal Coloring

Normal Map Operation

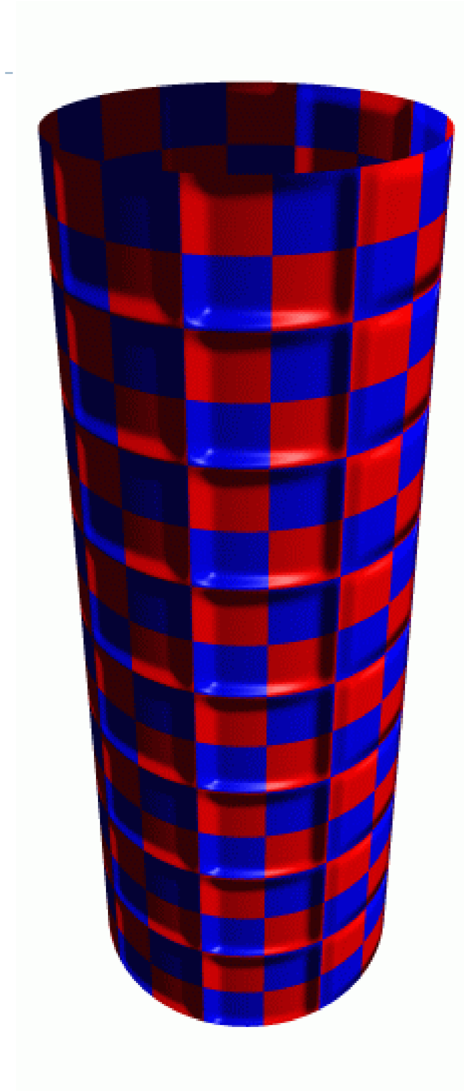


For each pixel, determine the normal from a texture image. Use that to compute the color.

What's Missing?

- There are no bumps on the silhouette of a bump or normal-mapped object

→ Displacement Mapping



What Next?

Graphics Courses

- CSE 167: Introduction to CG (Schulze, Ramamoorthi) – Fall and Winter
- CSE 165: 3D User Interfaces (Schulze) – Winter
- CSE 190: Advanced CG (Ramamoorthi) – Winter
- CSE 168: Rendering (Wann Jensen) – Spring
- CSE 169: Animation (Rotenberg) – Spring
- CSE 190: Virtual Reality (Schulze) – Spring
- CSE 198/199: Independent Study Projects

Computer Graphics State of The Art

- ACM SIGGRAPH Asia 2017 Technical Papers (3:28)
  - https://www.youtube.com/watch?v=3OGKh_9Rj_8
- 2017 Features | Unreal Engine (3:40)
  - https://www.youtube.com/watch?v=WC6Xx_jLXmg
- 2017 Student Reel | Unreal Engine (2:40)
  - https://www.youtube.com/watch?v=h4PAtDoYFaE

Good luck with your final projects!