Project #3

Ray Tracing

Produced by PStill written by Frank Siegert, http://www.wizards.de/pstill.html
This message will go away in the registered version - see 'registration.pdf'

Given Files:

main.c

- Parses command line arguments (input .ray file, output .bmp file, width, height, recursion limit, contribution threshold)
- Call RayTrace(args…)
- Saves image returned by RayTrace

bmp.c, bmp.h

- Similar to Project #1, but no alpha
- Provides Image functions (ImageNew, ImageFree, Image Copy, ImageGet/SetPixel)
- Provides saving and loading of .bmp into Image (BMPReadImage, BMPWriteImage)
- Called by .ray parser to load textures…

raydata.c, raydata.h

- Definitions for our data types
- Vector/Matrix algebra functions/macros

Produced by PStill written by Frank Siegert, http://www.wizards.de/pstill.html
This message will go away in the registered version - see ’registration.pdf’

parser.c, parser.h

- Contains a function for loading .ray files
- Takes pointers to data structures (camera, environment) and pointers to store lists of data structures into (lights, materials, textures)
- Loads the .ray file, fills the structures, mallocs and fills the lists
- Generates the node tree of objects in the scene (which have pointers to the appropriate material, textures…)
- Returns the root node of the tree

ray.c ray.h

- Contains main function RayTrace
  - Allocate an Image
  - Call the parser to load the tree and data
  - Recursively walks the tree and store the cummulative object/world and world/object matrices with the objects
  - Walks the tree and text dumps the tree to the screen
  - Free up memory and return image

Produced by PStill written by Frank Siegert, http://www.wizards.de/pstill.html
This message will go away in the registered version - see 'registration.pdf'

cube.c, cube.h, sphere.c, sphere.h, cylinder.c, cylinder.h, cone.c, cone.h, triangle.c, triangle.h…

- Contain primitive object specific data type
- Contain member functions for the specific primitive objects

Produced by PStill written by Frank Siegert, http://www.wizards.de/pstill.html
This message will go away in the registered version - see 'registration.pdf'

File Format (.ray)

Example:

```
#camera 0.0 0.0 4.0   0.0 0.0 -1.0   0.0 1.0 0.0   .523
#background 0.0 0.0 0.0
#ambient 0.1 0.1 0.1
#light_num 3
#light_point 0.8 0.8 0.8 -10.0 10.0 10.0  1.0 0.0 0.0
#light_spot 1.0 1.0 1.0   10.0 10.0 10.0  -0.5773 -0.5773 -0.5773  1.0 0.0 0.0  0.231 0.5
#light_dir  0.3 0.3 0.3  0.0 -0.707 -0.707
#texture_num 1
#texture test_texture.bmp
#material_num 5
#material   0.2 0.0 0.0   1.0 0.0 0.0  1.0 1.0 1.0   0.0 0.0 0.0  0.6 0.0 1.4  -1 !!
#material   0.0 0.2 0.0   0.0 1.0 0.0  1.0 1.0 1.0   0.0 0.0 0.0  0.4 0.0 1.5  -1 !!
#material   0.0 0.0 0.2   0.0 0.0 1.0  1.0 1.0 1.0   0.0 0.0 0.0  0.2 0.0 1.6  -1 !!
#material   0.2 0.2 0.0   0.7 0.7 0.1  1.0 0.9 1.0   0.0 0.0 0.0  0.9 0.5 1.4  -1 !!
#material   0.2 0.2 0.2   1.0 1.0 1.0  1.0 1.0 1.0   0.0 0.0 0.0  0.1 0.0 1.4  0 !!
#group_begin  1.0 0.0 0.0 0.0   0.0 1.0 0.0 0.0  0.0 0.0 1.0 0.0  0.0 0.0 0.0 1.0
   #shape_sphere  0  0.0 0.0 0.0  1.0
   #group_begin  1.0 0.0 0.0 -3.0   0.0 1.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 1.0
      #shape_box  1  0.0 0.0 0.0  1.0 1.1 1.2
      #group_begin  1.0 0.0 0.0 0.0   0.0 1.0 0.0 3.0  0.0 0.0 1.0 0.0  0.0 0.0 0.0 1.0
         #shape_cylinder  2  0.0 0.0 0.0  1.0 1.4
         #shape_cone  3  0.0 -2.1 0.0  1.71 0.62
      #group_end
      #group_begin  1.0 0.0 0.0 0.0   0.0 1.0 0.0 -3.0  0.0 0.0 1.0 0.0  0.0 0.0 0.0 1.0
        #shape_cone  3  0.0 0.0 0.0  1.21 1.32
      #group_end
   #group_end
   #group_begin  1.0 0.0 0.0 -3.0   0.0 1.0 0.0 0.0  0.0 0.0 1.0 0.0  0.0 0.0 0.0 1.0
      #shape_triangle  4  0.0 -1.0 0.0  1.0 1.0 -0.2  -1.0 0.85 0.11   0.0 0.0 1.0  0.0
                                      0.0 1.0  0.0 0.0 1.0   0.5 0.0  1.0 1.0  0.0 1.0
   #group_end
#group_end
```

Syntax:

camera *px py pz dx dy dz ux uy uz ha*
background *r g b*
ambient *r g b*
light_num *n*
light_point *r g b px py pz ca la qa*
light_spot *r g b px py pz dx dy dz ca la qa sc sd*
light_dir *r g b dx dy dz*

Produced by PStill written by Frank Siegert, http://www.wizards.de/pstill.html
This message will go away in the registered version - see 'registration.pdf'

texture_num *n*
texture *filename*

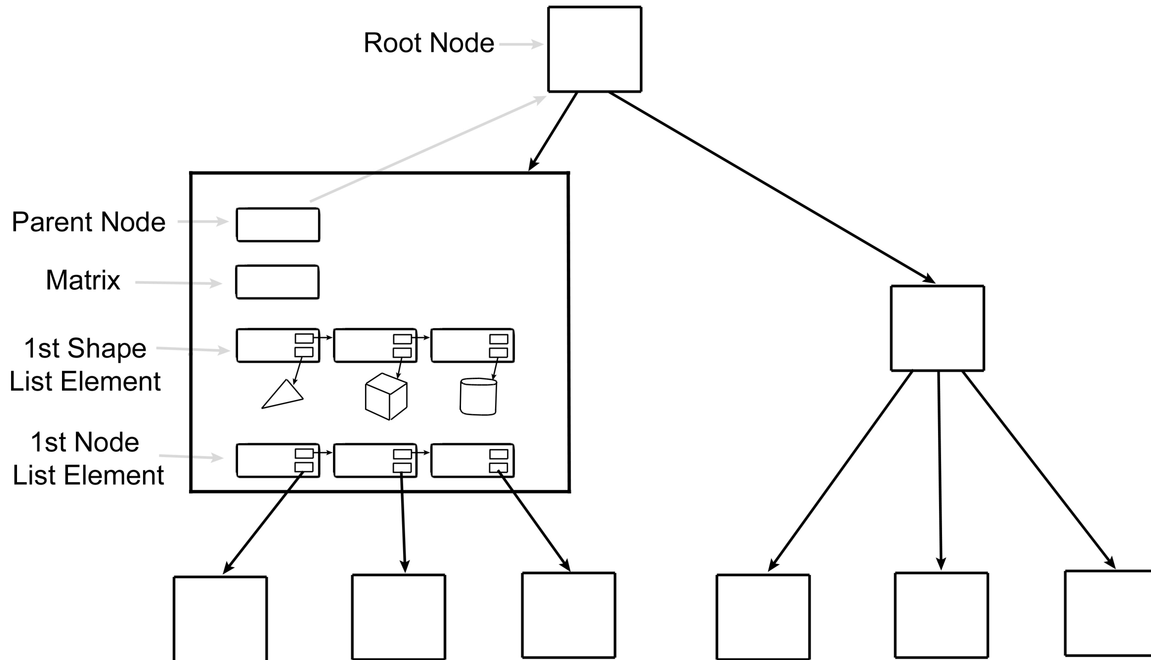
material_num *n*
material *ar ag ab dr dg db sr sg sb er eg eb*
ks kt ir tn !string!

group_begin *m11 m12 m13 m14 m21 m22 m23 m24*
m31 m32 m33 m34 m41 m42 m43 m44
group_end

shape_sphere *m cx cy cz r*
shape_box *m cx cy cz lx ly lz*
shape_cylinder *m cx cy cz r h*
shape_cone *m cx cy cz r h*
shape_triangle *m v1x v1y v1z v2x v2y v2z v3x v3y v3z*
n1x n1y n1z n2x n2y n2z n3x n3y n3z
t1s t1t t2s t2t t3s t3t

Produced by PStill written by Frank Siegert, http://www.wizards.de/pstill.html
This message will go away in the registered version - see 'registration.pdf'

Object Tree Structure




```
typedef struct TreeNode {
    Mat xform;
    TreeNode *parent_node;
    PrimListElement first_prim_list_element;
    TreeNodeListElement first_child_node_list_element;
} TreeNode;

typedef struct TreeNodeListElement {
    TreeNode *child_node;
    TreeNodeListElement *next_child_node_list_element;
} TreeNodeListElement;

typedef struct PrimListElement {
    Prim *prim;
    PrimListElement *next_prim_list_element;
} PrimListElement;
```

Produced by PStill written by Frank Siegert, http://www.wizards.de/pstill.html
This message will go away in the registered version - see 'registration.pdf'

Primitive Objects

```
typedef struct Prim {
        void *info;
        struct PrimProcs *procs;
        struct Material *material;
        Mat objectToWorld;
        Mat worldToObject;
} Prim;

typedef struct PrimProcs {
        void (*read)(Prim *, const char *);
        char *(*name) (Prim *);
        void (*print) (Prim *);
        int (*intersect) (Ray *, Prim *, Isect *);
        void (*normal) (Isect *);
        void (*texturecoord) (Isect *);
        void (*objectcoord) (Isect *);
        void (*destroy)(Prim *);
} PrimProcs;

typedef struct Material {
        Color ambient;
        Color diffuse;
        Color specular;
        Color emissive;
        Flt kspec;
        Flt ktran;
        Flt refind;
        Tex *tex;
        char foo[FOO_STRING_BUF_SIZE];
} Material;

typedef struct Tex {
        char filename[TEX_FNAME_BUF_SIZE];
        Image *img;
} Tex;
```

Produced by PStill written by Frank Siegert, http://www.wizards.de/pstill.html
This message will go away in the registered version - see ’registration.pdf’

Specific Objects

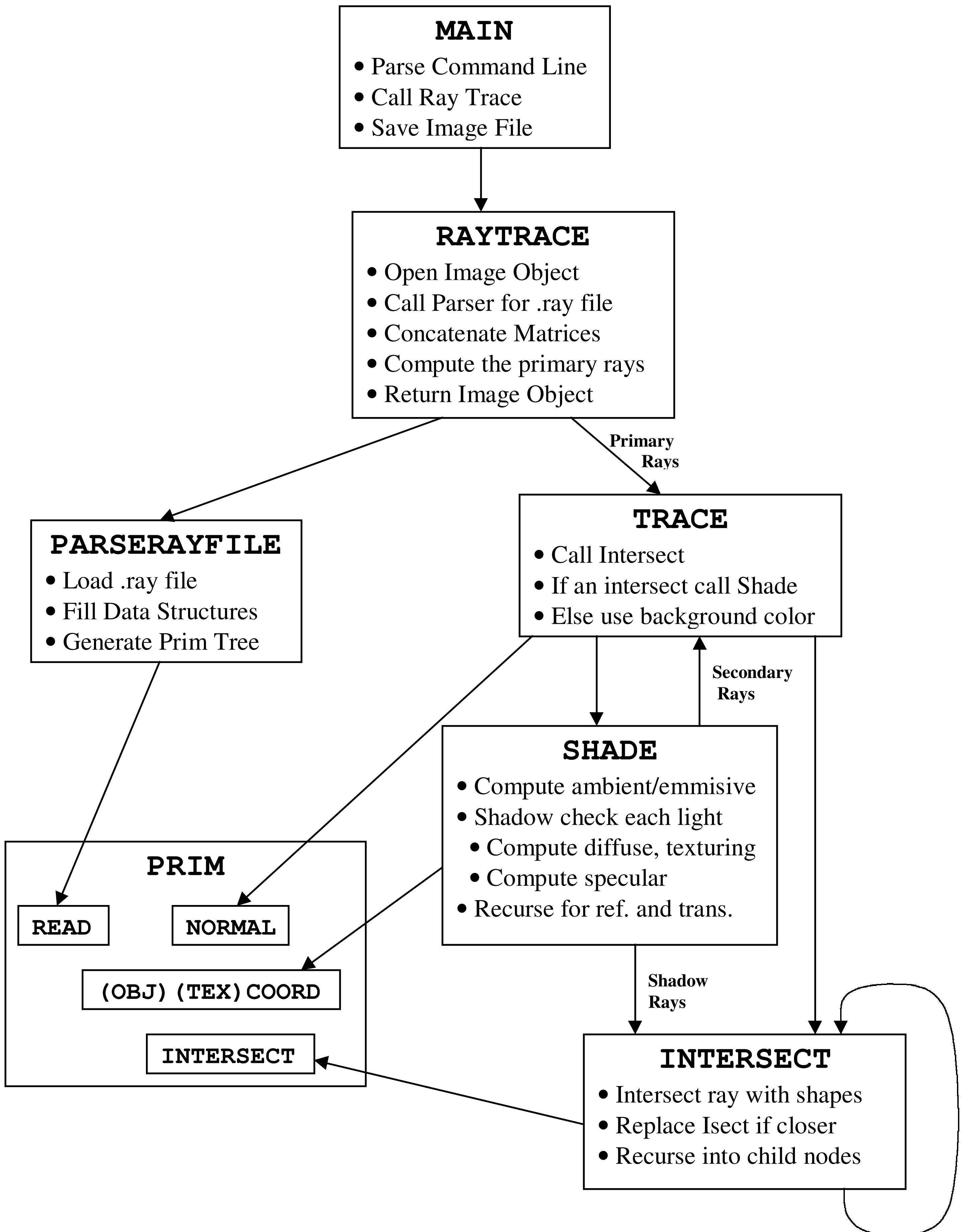
```
typedef struct Sph {
        Point cent;
        Flt rad;
} Sph;

void SphRead (Prim *prim, const char *stuff);
     Sph *s;
     s = malloc(sizeof(Sph));
     sscanf(stuff," #shape_sphere %*d %lg %lg %lg %lg",
          &s->cent[0], &s->cent[1], &s->cent[2], &s->rad);
     prim->info = (void *)s;
}

char * SphName (Prim *prim) return "sphere";

void SphPrint (Prim *prim) {
     Sph *s;
     s = (Sph *)prim->info;
     printf("SPHERE: C(%g, %g, %g) R=%g\n",
          s->cent[0], s->cent[1], s->cent[2], s->rad);
}

int SphIntersect (Ray *ray, Prim *prim, Isect *hit) {
     //Compute intersection(s)
     return numOfHits;
}
void SphNormal (Isect *hit) {  }

void SphTexCoord (Isect *hit) {  }

void SphObjCoord (Isect *hit) {  }

void SphRemove (Prim *prim) {
     Sph *s;
    s = (Sph *)prim->info;
    free(s);
}
PrimProcs SphProcs = { SphRead, SphName, SphPrint,
SphIntersect, SphNormal, SphTexCoord, SphObjCoord,
SphRemove};
```

Produced by PStill written by Frank Siegert, http://www.wizards.de/pstill.html
This message will go away in the registered version - see 'registration.pdf'

Other Data Types

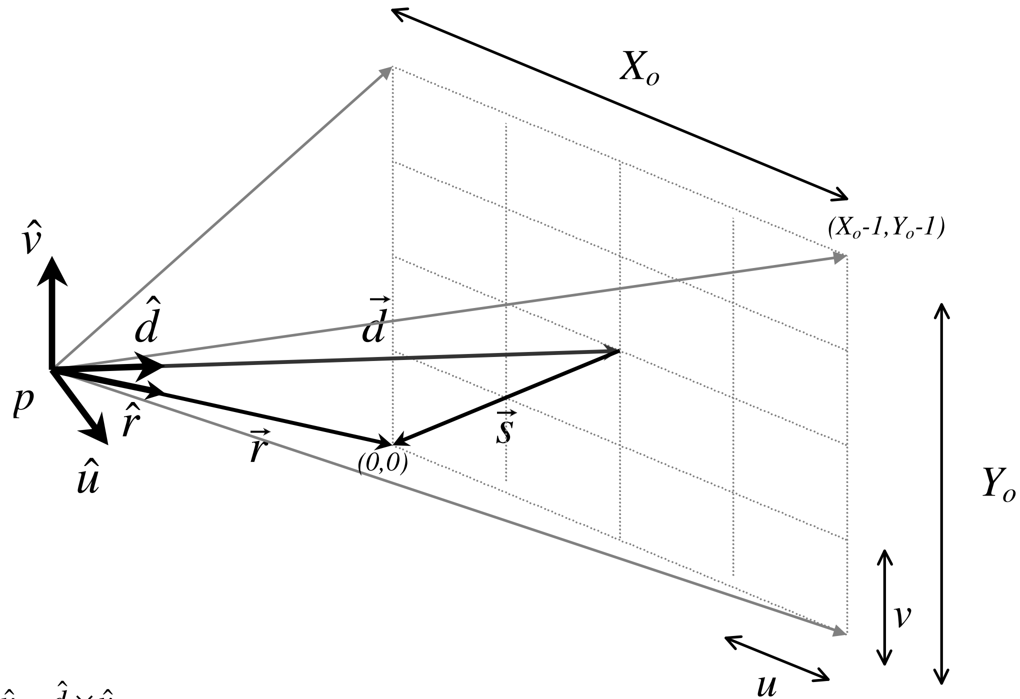
```
typedef struct Camera {
        Flt heightAngle;
        Flt aspectRatio;
        Point position;
        Dir direction;
        Dir up;
} Camera;


typedef struct Environ {
        Color ambient;
        Color background;
} Environ;


typedef struct Light {
        int type;
        Color color;
        Point location;
        Dir direction;
        Flt constAtten;
        Flt linearAtten;
        Flt quadAtten;
        Flt cutOffAngle;
        Flt dropOffRate;
} Light;

typedef struct Isect {
        Flt t;
        Prim *prim;
        Material *material;
        Point location;
        Dir norm;
        Point texcoord;
        Point objcoord;
} Isect;
```

Produced by PStill written by Frank Siegert, http://www.wizards.de/pstill.html
This message will go away in the registered version - see 'registration.pdf'



Produced by PStill written by Frank Siegert, http://www.wizards.de/pstill.html
This message will go away in the registered version - see 'registration.pdf'

Casting Rays:



$$\hat{u} = \hat{d} \times \hat{v}$$

$$u = \frac{2d \cdot \tan\theta_W}{X_O} = \frac{2 \cdot \tan\theta_W}{X_O}$$

$$v = \frac{2d \cdot \tan\theta_H}{Y_O} = \frac{2 \cdot \tan\theta_H}{Y_O}$$

$$\vec{s} = (X - \frac{X_O}{2})u\hat{u} + (Y - \frac{Y_O}{2})v\hat{v}$$

$$\vec{r} = \vec{d} + \vec{s} = d\hat{d} + \vec{s} = \hat{d} + \vec{s}$$

$$\vec{r} = \hat{d} + (X - \frac{X_O}{2})\frac{2 \cdot \tan\theta_W}{X_O}\hat{u} + (Y - \frac{Y_O}{2})\frac{2 \cdot \tan\theta_H}{Y_O}\hat{v}$$

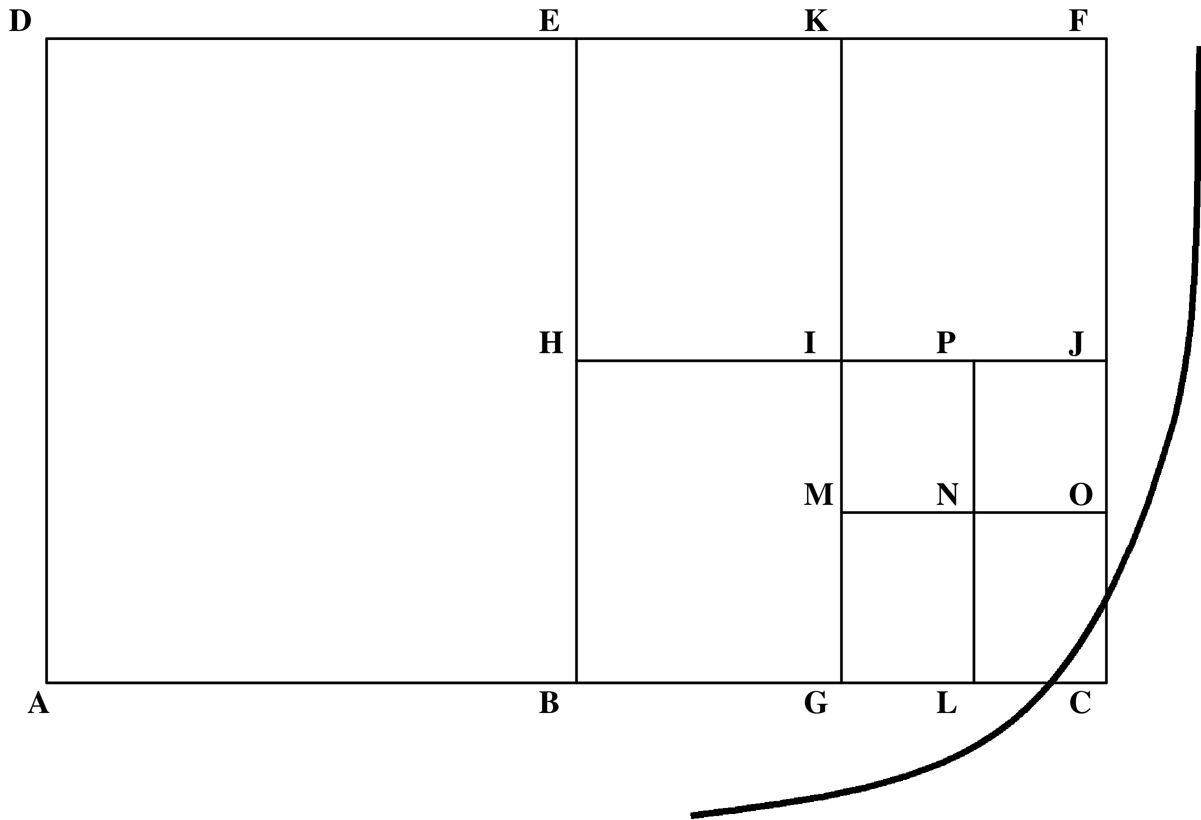
$$\hat{r} = \vec{r} \Big/ \left|\vec{r}\right|$$

$$ray = p + t \cdot \hat{r}$$

Produced by PStill written by Frank Siegert, http://www.wizards.de/pstill.html
This message will go away in the registered version - see 'registration.pdf'

Antialiasing via Oversampling

Adaptive Oversampling



$$p_1 = \frac{1}{4}[A + B + D + E]$$

$$p_2 = \frac{1}{4}[B + C + E + F] \rightarrow \frac{1}{4}\left[\frac{B+G+H+I}{4} + \frac{G+C+I+J}{4} + \frac{H+I+E+K}{4} + \frac{I+J+K+F}{4}\right]$$

$$\rightarrow \frac{1}{4}\left[\frac{B+G+H+I}{4} + \frac{1}{4}\left(\frac{G+L+M+N}{4} + \frac{L+C+N+O}{4} + \frac{M+N+I+P}{4} + \frac{N+O+P+J}{4}\right) + \frac{H+I+E+K}{4} + \frac{I+J+K+F}{4}\right]$$

Produced by PStill written by Frank Siegert, http://www.wizards.de/pstill.html
This message will go away in the registered version - see 'registration.pdf'

Intersect() – Brute Force

- Recursive function to walk object tree
- Attempt to intersect all objects in node
- Recursively call Trace() on all child nodes

Pseudo Code

```
Intersect(Ray *ray, TreeNode *node, Isect *isect){

     Isect testIsect = new Isect;
     int returnValue=0;

     /* Loop i over Prims in this node */
        if (prim[i]->intersect(ray,prim[i],testIsect)==1)
          if ((testIsect->t < isect->t)||(Isect->t < 0)){
                /* Copy testIsect into isect */
                /* Assume prim[i]->intersect gives t>0 */
                returnValue=1;
          }
     /*End loop*/


     /* Loop i child TreeNodes in this node */
        if (Intersect(ray,child[i],isect)==1)
             returnValue = 1;
     /*End loop*/

     delete testIsect;

     return returnValue;
}
```

Produced by PStill written by Frank Siegert, http://www.wizards.de/pstill.html
This message will go away in the registered version - see 'registration.pdf'

Shade

We augment the model in the handout:

$$Color = M_E + M_A L_G T_C + \sum_{i-Lights} L_i \left[M_D T_C (\hat{n} \cdot \hat{L}) + M_S (\hat{d} \cdot \hat{r})^{128 \cdot kSpec} \right] \cdot F_{SPOT} \cdot F_{ATTEN} +$$
$$kSpec \cdot M_S \cdot Trace(P + t\hat{r}) + kTrans \cdot M_S \cdot Trace(P + t\hat{t})$$

M_X – Material colors (Emissive, Ambient, Diffuse, Specular)
T_C – Texture color (at a given intersection point)
L_G – Global ambient light color
L_i – Diffuse/specular light color (for i^{th} light in sum)

P – Intersection point
$\underline{n}$ – Unit surface normal
$\underline{L}$ – Unit vector from isect point to light
$\underline{d}$ – Unit direction vector of intersection ray
$\underline{r}$ – Unit direction vector of reflection ray
t – Unit direction vector of transmitted ray

kSpec – Material specular reflection coeff. and shininess factor
kTrans - Material transmission coefficient

F_{SPOT} – Spot light factor = 0 outside cone or $(L \cdot d_{spot})^{128\ dropOffRate}$
F_{ATTEN} – Attenuation factor = $(a_c + a_l\, d + a_q\, d^2)^{-1}$.

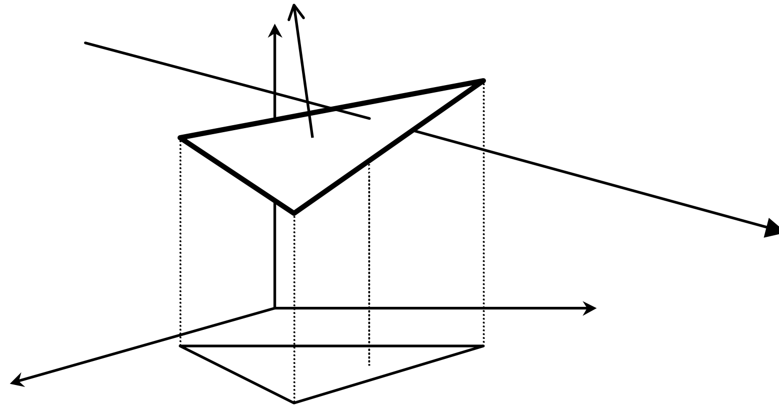
We can also deviate by:

- Flip normals inside shade (enter/exit)
- Assume one material is always air (n=1)
- Soft shadows (area lights)

Produced by PStill written by Frank Siegert, http://www.wizards.de/pstill.html
This message will go away in the registered version - see ’registration.pdf’

Specific Intersections

Triangle

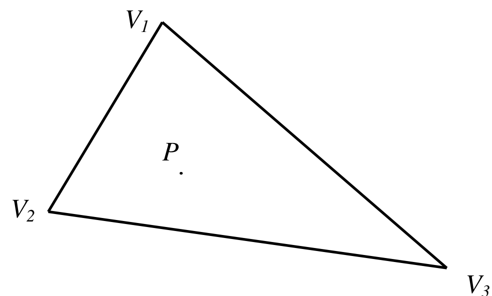


Three step process:

- Intersect ray with plane (3D)
- Project triangle and point into an axis aligned plane
- Check if point lies in triangle (2D) using barycentric coordinates (α,β,γ):

$$P = \alpha V_1 + \beta V_2 + \gamma V_3$$

$$\alpha + \beta + \gamma = 1$$
$$0 \le \alpha, \beta, \gamma \le 1$$



Produced by PStill written by Frank Siegert, http://www.wizards.de/pstill.html
This message will go away in the registered version - see 'registration.pdf'

Intersecting Rays with Planes:

Plane $ax + by + cz = d$

Normal $\hat{n} = (a, b, c)$

Ray $P + t\hat{d} = (p_x, p_y, p_z) + t(d_x, d_y, d_z)$

Substitute $a(p_x + td_x) + b(p_y + td_y) + c(p_z + td_z) = d$

$$P \cdot \hat{n} + t\hat{d} \cdot \hat{n} = d$$

$$t = \frac{d - P \cdot \hat{n}}{\hat{d} \cdot \hat{n}}$$

Pseudo Code:

```
int TriIntersect (Ray *ray, Prim *prim, Isect *hit) {

Flt u0,u1,u2,v0,v1,v2,a,b;
Tri *tr;
int axis1, axis2;

tr = (Tri *)prim->info;

/* Compute temp = d dot n */
/* if temp == 0 ray is parallel to plane so return 0 */
/* else finish computing hit->t */
/* if t<RAYEPS return 0 */

RayPoint(ray,hit->t,hit->location);

/* Find axis1, axis2 - The indices of shortest
     components of the triangle's true normal */
u0 = hit->location[axis1] - tr->vert[0][axis1];
v0 = hit->location[axis2] - tr->vert[0][axis2];
u1 = tr->vert[1][axis1] - tr->vert[0][axis1];
v1 = tr->vert[1][axis2] - tr->vert[0][axis2];
u2 = tr->vert[2][axis1] - tr->vert[0][axis1];
v2 = tr->vert[2][axis2] - tr->vert[0][axis2];
```

Produced by PStill written by Frank Siegert, http://www.wizards.de/pstill.html
This message will go away in the registered version - see 'registration.pdf'

```
    if (u1 == 0) {
        if (((g=u0/u2)>=0.0)&&(g<=1.0)) {
            if (((b=(v0-g*v2)/v1)<0.0)||((b+g)>1.0)) return 0;
        }
        else return 0;
    }
    else {
        if (((g=(v0*u1-u0*v1)/(v2*u1-u2*v1))>=0.0)&&(g<=1.0)) {
            if (((b=(u0-g*u2)/u1)<0.0)||((b+g)>1.0)) return 0;
        }
        else return 0;
    }

    hit->prim = prim;
    hit->medium = prim->surf;
    VecSet(hit->norm,(1-b-g),b,g);
        VecSet(hit->texcoord,(1-b-g),b,g);


    return 1;
}
```

Notes:

- Move 2D origin to vertex[0] and solve for beta and gamma (2 eqn's, 2 unknowns)
- Now normals and texture coordinates are trivially interpolated.
- You might modify the Tri struct and TriRead function to have/store some things
- What about tranformation matrices?!?!?

Produced by PStill written by Frank Siegert, http://www.wizards.de/pstill.html
This message will go away in the registered version - see 'registration.pdf'

Transformation Matrices

- Most intersections tests are *much* easier in object coordinates
- Transform ray into object space
- Intersect in object space
- Transform intersection points and normals back into world coordinates

BE VERY CAREFUL:

- Points, directions, and normals DON'T all transform the same way
- ALWAYS renormalize directions/normals

$$P = [x \quad y \quad z \quad 1]^T \qquad P' = MP$$

$$\vec{D} = [x \quad y \quad z \quad 0]^T \qquad \vec{D}' = M\vec{D}$$

$$\hat{N} = [x \quad y \quad z \quad d]^T \qquad \hat{N}' = (M^{-1})^T \hat{N}$$

Using our functions:

```
VecMatrixMult(P,M,P')
VecMatrixMultDirection(D,M,D')
VecMatrixMultTrans(N,M_inverse,N')
```

Produced by PStill written by Frank Siegert, http://www.wizards.de/pstill.html
This message will go away in the registered version - see 'registration.pdf'

Other Ray/Object Intersections

- Cylinder (like sphere)

$$(x - x_0)^2 + (z - z_0)^2 = r^2$$

- Cone (like sphere)

$$(x - x_0)^2 + (z - z_0)^2 = \left[\frac{r}{h} \left(y - \frac{h}{2} - y_0 \right) \right]^2$$

- Cube

  Lots of methods… Maybe slabs?

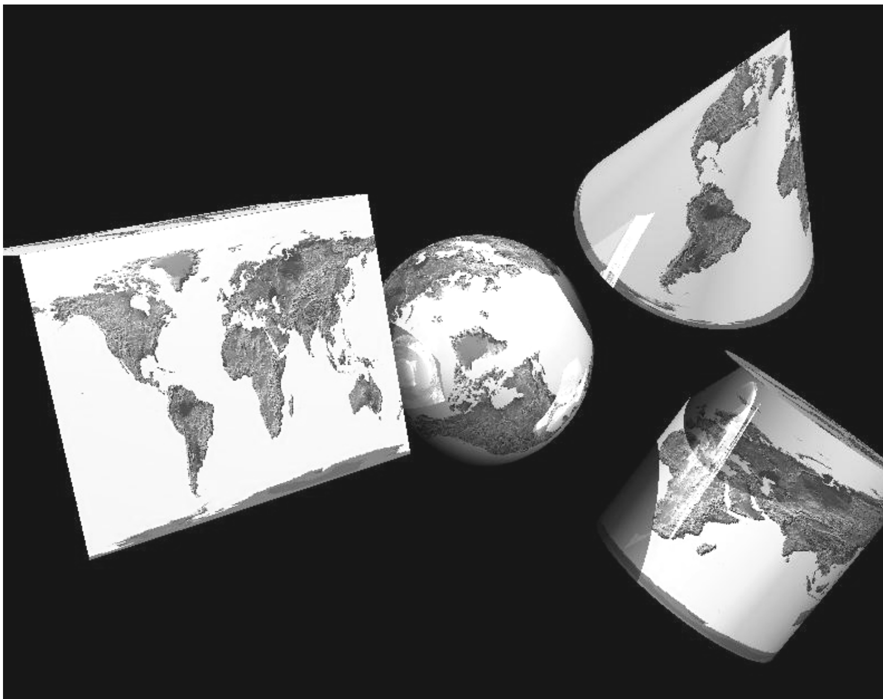
Plenty of nice algorithms here:

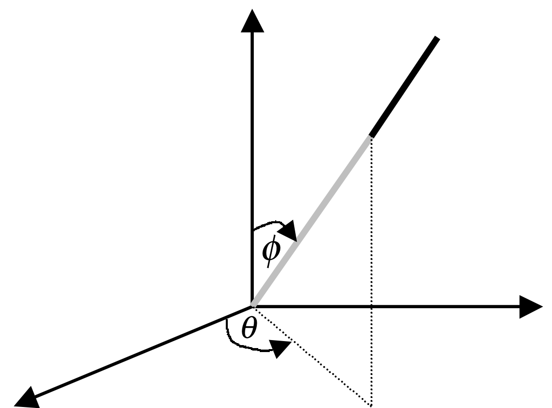
```
ftp://ftp.princeton.edu/pub/
                    Graphics/GraphicsGems/
```

Produced by PStill written by Frank Siegert, http://www.wizards.de/pstill.html
This message will go away in the registered version - see 'registration.pdf'

Texturing Mapping (2D)

- Assume that we always wrap in s and t
- (s,t) = (0,0) is in the upper left
- Co-ordinate directions of s and t correspond to 'right' (x) and 'down'(y).
- You only need to worry about triangles.
- Bonus: Setup texture mapping for spheres, cones, cylinders, cubes.
- Map s,t (0 - 1) to θ ($0 - 2\pi$), ϕ ($0 - \pi$) coordinates for spheres, for instance.
- Use ./rayview to see proper mapping.





Produced by PStill written by Frank Siegert, http://www.wizards.de/pstill.html
This message will go away in the registered version - see 'registration.pdf'

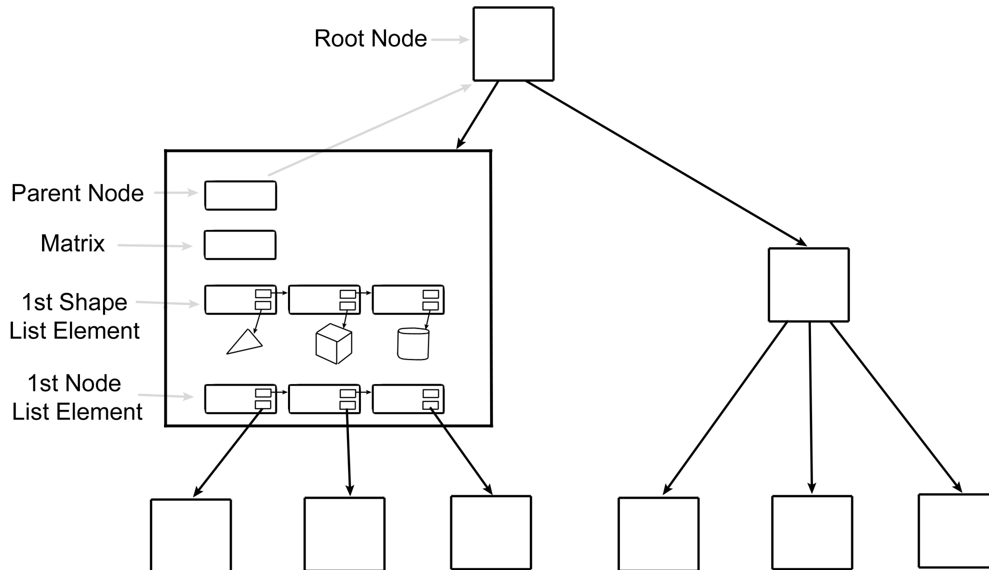
3-D Texturing

- Develop functions of coordinates (x,y,z) that describe a 3-D material
- Evaluate the function at the intersection points in object coordinates.
- Modulate the diffuse ambient color with this scalar (or maybe three different channels)
- Bump mapping is easy – Perturb the normal at the intersection using the partial derivatives of the function (slopes)
- One good possibility (if you have time to kill): http://freespace.virgin.net/hugo.elias/models/m_perlin.htm

Produced by PStill written by Frank Siegert, http://www.wizards.de/pstill.html
This message will go away in the registered version - see 'registration.pdf'

Acceleration Methods:

Hierarchical Bounding Boxes:



Octrees:

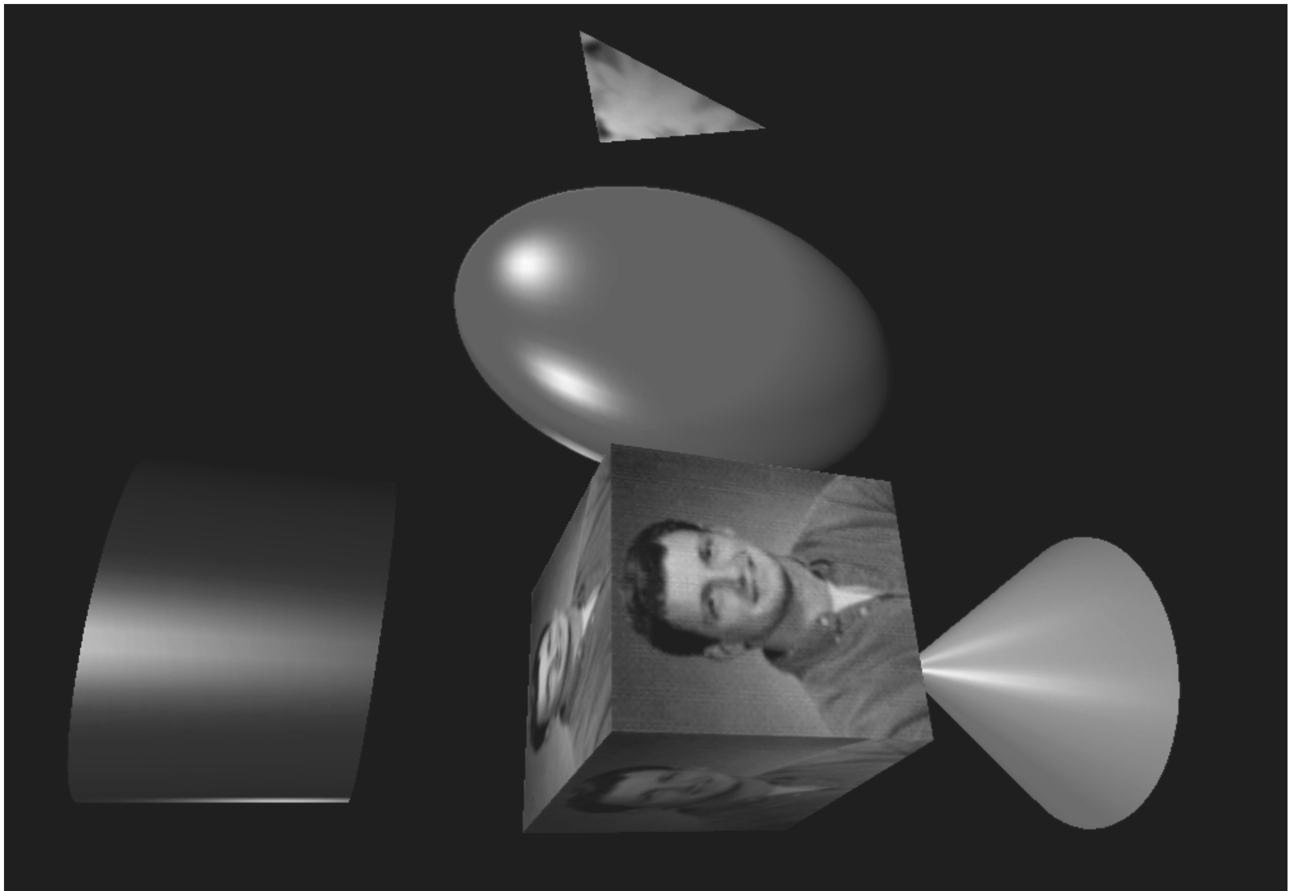
- Partition space into 8 octants
- Sub-partition until each octant has less than the desired number of objects
- Ray trace by intersecting only the octants you pierce

BSP Trees: (~binary octrees)

Produced by PStill written by Frank Siegert, http://www.wizards.de/pstill.html
This message will go away in the registered version - see 'registration.pdf'

Testing:

- Create simple .ray files that test you incremental functional implementation.
- Use *rayview* to view .ray files in OpenGL (shows everything but shadows, transparency, reflections…)



Produced by PStill written by Frank Siegert, http://www.wizards.de/pstill.html
This message will go away in the registered version - see 'registration.pdf'