DATA-ORIENTED DESIGN (for math)
@Mike-actor
anti-math talk?
WHAT IS DATA-ORIENTED?
Practical philosophy not methodology.
Way to approach Problem Solving.
Basic Principles
Know the data
Know the context
Data > Code
Nothing new.
abandoning crap.
The only purpose of any code is to transform data.
PROGRAMMER IS FUNDAMENTALLY RESPONSIBLE FOR THE DATA, NOT THE CODE.
ONLY WRITE CODE THAT HAS DIRECT, PROVABLE VALUE.

i.e. TRANSFORMS DATA IN MEANINGFUL WAY
CODE IS JUST A DATA DESCRIPTION.

(AND IS JUST DATA.)
UNDERSTAND THE DATA TO UNDERSTAND THE PROBLEM.
There is no ideal abstract solution to the problem.
You can't "future proof"
Common Misconceptions
All about SIMD
only about performance
OPTIMIZABILITY

CAN BE VS.

IMPOSSIBLE.
- very

Harder to understand, maintain, etc.
very likely,
you already
do it (to
some degree)
But it's not the exception
Lights
Shaders
particles
particles

animation
Culling.
What do we take away from these examples?
A generic solution does not exist.
A generic solution cannot exist.
all solutions have either explicit or implicit context.
too often implicit & unknown
Implicit Limits

10? 100? 1000? 15,000?
Simple C++ HDS class definition

struct HalfEdge
{
  HalfEdge *start;
  HalfEdge *endpt;
  HalfEdge *next;
  HalfEdge *opposite;
  HalfEdgeFace *face;
  void *userData;
  unsigned char marker;
  unsigned char index;
};

struct HalfEdgeVert
{
  HalfEdge *halfEdge;
  unsigned char marker;
  int index;
};

{ void *userData;
  unsigned char marker;
};
Architectures

Sustainable

Green

Cell
TRY TO SOLVE ALL PROBLEMS, SOLVE VERY FEW.
WHAT CASE TO SOLVE FOR?
WORST?
BEST?
MOST COMMON?
EDGE?
TOTALY
UNKNOWN
CASES?
MATH Lib Influence on Code.
BAD

“CONTEXT
FREE”
Squirrel -

"best things about
parametric..."

without memory
or context"
SOMEONE HAS TO SAY

"BUT THAT'S ONLY USEFUL FOR GAMES!"
"THEY" or "USER"

MENTAL MODEL = PROBLEM
kamiphish - C++, pasted 1 hour ago:

```
# define MAX_PLANES 6

enum { NONE, IN, OUT };

struct Sphere
{
    float x, y, z;
    float r;
```

Typical.
class Object
{
public:
    virtual void onVisible(Culler&) = 0;
    const Sphere& WorldBound() const;
};

int WhichSide(const Plane& p, const Sphere& s)
{
    float d = p.x * s.x + p.y * s.y + p.z * s.z + p.c;
    if (d <= -s.r)
        return OUT;
    else if (d >= s.r)
        return IN;
    else
        return NONE;
}

class Culler
{
    Plane m_planes[MAX_PLANES];
    unsigned int m_planeState;
}
class Culler
{
public:
    virtual void onVisible(Culler&) = 0;
    const Sphere& WorldBound() const;
};

int WhichSide(const Plane& p, const Sphere& s)
{
    float d = p.x * s.x + p.y * s.y + p.z * s.z + p.c;
    if (d <= -s.r) return OUT;
    else if (d >= s.r) return IN;
    else return NONE;
}

class Culler
{
    Plane m_planes[MAX_PLANES];
    unsigned int m_planeCount;
};
OH YEAH BIC WE THINK LOGICAL RESULT "CAN'T" BE FLOAT.
class Object
{
    public:
        virtual void onVisible(Culler&) = 0;
    const Sphere& WorldBound() const;
};

int WhichSide(const Plane& p, const Sphere& s)
{
    float d = p.x * s.x + p.y * s.y + p.z * s.z + p.c;
    if (d <= -s.r)
        return OUT;
    else if (d >= s.r)
        return IN;
    else
        return NONE;
}

class Culler
{
    Plane m_planes[MAX_PLANES];
    unsigned int m_planes;

NOT TO MENTION CRAZY BRANCHES.
class Object
{
    public:
        virtual void onVisible(Culler&) = 0;
        const Sphere& WorldBound() const;
    
};

int WhichSide(const Plane& p, const Sphere& s)
{
    float d = p.x * s.x + p.y * s.y + p.z * s.z + p.c;

    if (d > 0.0)
        return 1;
    else if (d < 0.0)
        return -1;
    else
        return 0;

};

class Culler
{
    Plane m_plane;
    unsigned int m_last_who;

    public:
        Culler(Plane plane, unsigned int last_who);
};
class Object{
    public:
        virtual void onVisible(Culler&) = 0;
        const Sphere& WorldBound() const;
};

int WhichSide(const Plane& p, const Sphere& s){
    float d = p.x * s.x + p.y * s.y + p.z * s.z + p.c;
    if (d > 0.0) return 1;
    else if (d < 0.0) return -1;
    else return 0;
}

class Culler{
    Plane unsigned

public:;
unsigned plane CULL;
class Cull
{
int whichSide(const plane & p, const sphere & s);
void setPlane(const plane & p);
virtual void setVisible(Culler & c);

class Culler
{;
};
int WhichSide(const Plane& p, const Sphere& s)
{
    float d = p.x * s.x + p.y * s.y + p.z * s.z + p.c;
    if (d > 0.0)
        return 1;
    else if (d < 0.0)
        return -1;
    else
        return 0;
}

class Cull
{
    Plane unsigned;

    public:

    void Cull()
    {
    }
}

Don't build FUNC FOR < .01% case here as "GEMEAL" SOLUTION.
THE "GENERAL" CASE IS MANY
class Culler {
    Plane m_planes[MAX_PLANES];
    unsigned int m_planeState;

public:

    void CullInvisibleObjects(Object* object) {
        if (!m_planeState)
            object->onVisible(*this);
        else
            // Determine if the object is not visible bound to each culling plane.
else
    return NONE;

class Culler
{
    Plane m_planes[MAX_PLANES];
    unsigned int m_planeState;

public:
    void CullInvisibleObjects(Object* object)
    {
        if (!m_planeState)
        {
            object->invisible();
        }
        else
        {
            //
            //
        }
    }

    //
    //
}
IS THERE REALLY JUST ONE UNIQUE OBJECT IN THE APP?
Uh. No.
if (d <= -s.r)  
    return OUT;
else if (d >= s.r)  
    return IN;
else  
    return NONE;

class Culler

    Plane m_planes[MAX_PLANES];
    unsigned int m_planeState;

public:

    void CullInvisibleObjects(Object* object)  
    {
        if (!m_planeState)  
        {
            object->onVisible(*this);
        }
        else  
        {
            // Determine if the object  
            // bound to each culling pl
// Culler

Plane m_planes[MAX_PLANES];

unsigned int m_planestate;

public:
  void CullInvisibleObject;
  { if (!m_planestate)
    { object->onVisible();
    } else 
      { // Determine bound to each
        // plane and cull if
        // appropriate.
    } 

  }

  // (Note: I have not tested
  // the above code. The code
  // itself seems fine, but I
  // have not tested it. I
  // would appreciate it if
  // someone could test it and
  // let me know if it works.)

  A
class Culler;

class Object {
    public:
        virtual void onVisible(Culler&) = 0;
        const Sphere& WorldBound() const;
};

int WhichSide(const Plane& p, const Sphere& s) {
    float d = p.x * s.x + p.y * s.y + p.z;
    if (d <= -s.r) return OUT;
    return IN;
}

WAIT, LET ME BACK UP A MINUTE.
class Culler;

class Object
{
  public:
  virtual void onVisible(Culler&); = 0;
  virtual void onVisible() const;
};

const Sphere s = Sphere(0.1, 0.1, 0.1, 0.1);
First, you can sort by type. So you can make correct call.
class Culler;

class Object
{
 public:
  virtual void onVisible(Culler&) = 0;

  virtual ~WorldBound() const;

  int x;

  float y;

  float z;

  float w;

  float scaleX;

  float scaleY;

  float scaleZ;

  float scaleW;

  float skewX;

  float skewY;

  float skewZ;

  float skewW;

  float rotationX;

  float rotationY;

  float rotationZ;

  float rotationW;

  float translateX;

  float translateY;

  float translateZ;

  float translateW;

  float tintR;

  float tintG;

  float tintB;

  float tintA;

  float opacity;

  float is Transparent;

  float is Masked;

  float is Visible;

  float is BoundingBox;

  float is Frustum;

  float is Shadow;

  float is Camera;

  float is Refraction;

  float is Reflection;

  float is Light;

  float is Shadow;

  float is Camera;

  float is Refraction;

  float is Reflection;

  float is Light;
class Culler;

class Object
{
public:
  virtual void onVisible(Culler&) = 0;
  virtual void worldBound() const;
};
... although a "good" use is hard to come by.
unsigned int saveActivePlanes = m_planeState;

unsigned int i;
for (i = 0; i < MAX_PLANES; i++)
{
    int m = 1 << i;
    if (m_planeState & m)
    {
        int side = WhichSide(m);
        if (side == OUT)
        {
            // The object is on
            // side of the
            // plane.
            break;
        }
        if (side == IN)
        {
            // The object is in
            // so there is
            // plane.
            m_planeState &= m;
        }
    }
}

← ARCH!
AGAIN?
SORT THE
DATA!
SORT IT!
signed int saveActivePlanes = m_planeState;

signed int i;
(i = 0; i < MAX_PLANES; i++)
int m = 1 << i;
if (m_planeState & m) {
    int side = WhichSide;
    if (side == OUT) {
        // The object is on the
        // side of the plane.
        break;
    }
    // The object is on the
    // side of the plane.
    // So there is no
    // plane.
    m_planeState &=

← KNOW THE DATA!
How MANY PLANE ARRANGE-
MENTS Do you
REALLY USE?
Then call correct func to xform that data (in group) no test. No branch.
unsigned int
for (i = 0; i < MAX_PLANES;
{
    int m = 1 << i;
    if (m_planeState & m)
        int side = WhichSide(m_planes[i], object->WorldBound);

        if (side == 2)
            // The object is not visible since it is on the
            // positive side of this plane type.

        // Note you clearly have >1 plane at
        // this pt.
unsigned int i;
for (i = 0; i < MAX_PLANES; i++) {
    int m = 1 << i;
    if (m_planeState & m)
    {
        int side = WhichSide(m_planes[i], object->WorldBound[k]);
        // The object is not visible since
        // it is on the wrong side of the plane.
        if (side == 3)
            // The object is not visible since
            // it is on the wrong side of the plane.

        But still calling indiv. func.
        So lame!
unsigned int i;
for (i = 0; i < MAX_PLANES; i++)
{
    int m = 1 << i;
    if (m_planeState & m)
    {
        int side = WhichSide(m_planes[i], object-);

        if (side == OUT)
        {
            // The object is not visible
            ...
        }
    }
}

↑
NOW WE'RE STUCK w/ THIS INT
unsigned int m = 1 << i;
if (m_planeState & m)
{
    int side = WhichSide(m_planes[i], object->WorldPosition, ...);
if (side == OUT)

   BASELACELY FORCING YOU TO BRANCH
int saveActivePlane

int i;
0; i < MAX_PLANES; i++)

m = 1 << i;
(m_PlaneState & m)

(int side = WhichSide(m_planes[i], object->WorldBound())

if (side == OUT)
{
    // The object is not visible since it is on the other plane.
    break;
}

if (side == IN)
{
    // The object is on this plane.
}
if (i == MAX_PLANES)
{
    object->onVisible(*this);
}
KILLING ICACHE FOR NO REASON.
CALL invisible LATER, on A WHOLE LIST.
function defined for

^ GHOST WRITES.
BAD FOR
CACHE. BAD
FOR THE SOUL.
public:
    virtual void onVisible(Culler&) = 0;
    const Sphere& WorldBound() const;

^}

while I'm here. References? Pointless cruft.
Mike Acton

#AltDevBlogADay #gamedev

@mike_acton
macton@gmail.com
http://www.linkedin.com/in/mikeacton