Bounding Volume Hierarchies

CS 6965 Fall 2011
Programmers' Day

From Wikipedia, the free encyclopedia

Programmers' Day (Russian: День программиста) is an international professional holiday, recognized in many technology companies and programming firms, that is celebrated on the 256th (0\times100th) day of each year (September 13 during common years and on September 12 in leap years). It is also officially recognized in Russia[1][2] and observed in several other countries including Poland, the United States, United Kingdom, Germany, France, China, Guatemala, and Croatia.[Citation needed]

Contents

1 Celebration
2 Official recognition
3 See also
4 References
5 Sources

Celebration

The number 256 (2^8) was chosen because it is the number of distinct values that can be represented with an eight-bit byte — a value well known to programmers.[3] Starting from zero, the 256th value represented by a sequential permutation of 8 bits is unsigned integer 256 or hexadecimal 0xff or binary 0b1111111. 256 is the highest power of two that is less than 365, the number of days in a common year.

Official recognition

This particular day was proposed by Valentin Balt, an employee of Parallel Technologies web design company. As early as 2002 he tried to gather signatures for a petition to the Government of Russia to recognize the day as the official programmers' day.[4]

On July 24, 2009, the Ministry of Mass Communications of Russia issued a draft of an executive order on a new professional holiday, Programmers' Day.[5][8]

On September 11, 2009, President of Russia Dmitry Medvedev signed the decree.[2][7]

See also

- World Information Society Day
- System Administrator Appreciation Day
Heightfield
C++ Operator?

#include <stdio.h>
int main()
{
    int x = 10;
    while( x --> 0 ) // x goes to 0
    {
        printf("%d ", x);
    }
}

stackoverflow.com/questions/1642028/what-is-the-name-of-this-operator
C++ Operator?

That's not an operator -->. That's two separate operators, -- and >. Your post is decrementing x and then comparing x and 0 with the > operator.

To better understand, the statement could be as follows:

while( (x--) > 0 )
Program 2
Backwards Lighting
Light Direction

Compute hit position \( \vec{P} = \vec{O} + t\vec{V} \)
Call primitive to get normal \( \vec{N} \) (normalized)
\[
\text{costheta} = \vec{N} \cdot \vec{V}
\]
\[
\text{if (costheta} < 0 \)
\quad \text{normal} = -\text{normal}
\]
Color light = scene.ambient * Ka
foreach light source
  get \( C_L \) and \( \vec{L} \)
  dist = \( ||\vec{L}||, \vec{L}_n = \frac{\vec{L}}{||\vec{L}||} \)
  \[
  \text{cosphi} = \vec{N} \cdot \vec{L}_n
  \]
  \[
  \text{if (cosphi > 0) }
  \quad \text{if (!intersect with } 0 < t < \text{dist) }
  \quad \text{light} += C_L \times (Kd \times \text{cosphi})
  \]
result = light * surface color

- \text{costheta} > 0
  - Means \( N \) and \( V \) are on opposite sides
  - Should flip the normal

- \text{costheta} < 0
  - Means \( N \) and \( V \) are on the same side
  - Keep the normal
Library Calls

• Show up as something like this:

  ERROR: undefined symbol: __dso_handle
  failed to add instruction: ORI
  line 54
  assembler returned an error, exiting

• This one was fixed by removing global variables

• Use #define or keep things locally
Memset

- We do have this one!
- Change this line: 
  ```bash
  @${LINKERDIR}/ln.py rt.s > ${TRAXOUTPUT}
  ```
- To this line: 
  ```bash
  @${LINKERDIR}/ln.py rt.s ${LIBDIR}/memset.s > ${TRAXOUTPUT}
  ```
Inheritance

- We can’t handle these in simhwrt yet
Calling Convention
void draw_line(const int& framebuffer, const int& width, const Vector& V1, const Vector& V2, const int& max_pixel)
{
    int x0 = V1[0];
    int x1 = V2[0];
    int y0 = V1[1];
    int y1 = V2[1];

    int dx = x0 - x1;
    if (dx < 0) dx = -dx;
    int dy = y0 - y1
    if (dy < 0) dy = -dy;

    int sx, sy;
    if (x0 < x1) sx = 1; else sx = -1;
    if (y0 < y1) sy = 1; else sy = -1;
    int err = dx - dy;
    int e2;
    while (x0 != x1 || y0 != y1)
    {

Common Problems

```c
int x0 = V1[0];
int x1 = V2[0];
int y0 = V1[1];
int y1 = V2[1];

trax_printi(x0); trax_printi(x1); trax_printi(y0); trax_printi(y1);

int dx = x0 - x1;
if (dx < 0) dx = -dx;
int dy = y0 - y1
if (dy < 0) dy = -dy;

int sx, sy;
if (x0 < x1) sx = 1; else sx = -1;
if (y0 < y1) sy = 1; else sy = -1;
int err = dx - dy;
int e2;

trax_printi(err);
```
int x0 = V1[0];
int x1 = V2[0];
int y0 = V1[1];
int y1 = V2[1];

trax_printi(x0); trax_printi(x1); trax_printi(y0); trax_printi(y1);

int dx = x0 - x1;
if (dx < 0) dx = -dx;
int dy = y0 - y1
if (dy < 0) dy = -dy;

int sx, sy;
if (x0 < x1) sx = 1; else sx = -1;
if (y0 < y1) sy = 1; else sy = -1;

trax_printi(dx); trax_printi(dy);

int err = dx - dy;
int e2;
Common Problems

```cpp
int x0 = V1[0];
int x1 = V2[0];
int y0 = V1[1];
int y1 = V2[1];

trax_printi(x0); trax_printi(x1); trax_printi(y0); trax_printi(y1);

int dx = x0 - x1;
if (dx < 0) dx = -dx;
int dy = y0 - y1
if (dy < 0) dy = -dy;

trax_printi(dx); trax_printi(dy);

int sx, sy;
if (x0 < x1) sx = 1; else sx = -1;
if (y0 < y1) sy = 1; else sy = -1;

int err = dx - dy;
int e2;
```
Common Problems

```
trax_printi(x0); trax_printi(x1); trax_printi(y0); trax_printi(y1);

int dx = x0 - x1;
if (dx < 0) dx = -dx; // dx = abs(dx)
int dy = y0 - y1
if (dy < 0) dy = -dy; // dy = abs(dy)

trax_printi(dx); trax_printi(dy);
```
Common Problems

```c
trax_printi(x0); trax_printi(x1); trax_printi(y0); trax_printi(y1);

int dx = x0 - x1;
if (dx < 0) dx = -dx;  // dx = abs(dx)
int dy = y0 - y1
if (dy < 0) dy = -dy;  // dy = abs(dy)

trax_printi(dx); trax_printi(dy);

// Write it a different way
int dx = x0 > x1 ? x0 - x1 : x1 - x0;
int dy = y0 > y1 ? y0 - y1 : y1 - y0;
```
Common Problems

Before

```c
void draw_line(const int& framebuffer, const int& width, const Vector& V1, const Vector& V2, const int& max_pixel)
{
    int x0 = V1[0];
    int x1 = V2[0];
    int y0 = V1[1];
    int y1 = V2[1];

    int dx = x0 - x1;
    if (dx < 0) dx = -dx;
    int dy = y0 - y1
    if (dy < 0) dy = -dy;

    int sx, sy;
    if (x0 < x1) sx = 1; else sx = -1;
    if (y0 < y1) sy = 1; else sy = -1;
    int err = dx - dy;
    int e2;
    while (x0 != x1 || y0 != y1)
    {
        // Code...
    }
}```

After

```c
void draw_line(const int& framebuffer, const int& width, const Vector& V1, const Vector& V2, const int& max_pixel)
{
    int x0 = V1[0];
    int x1 = V2[0];
    int y0 = V1[1];
    int y1 = V2[1];

    int dx = x0 > x1 ? x0 - x1 : x1 - x0;
    int dy = y0 > y1 ? y0 - y1 : y1 - y0;

    int sx, sy;
    if (x0 < x1) sx = 1; else sx = -1;
    if (y0 < y1) sy = 1; else sy = -1;
    int err = dx - dy;
    int e2;
    while (x0 != x1 || y0 != y1)
    {
        // Code...
    }
```
Bounding boxes

- **Box**: easy
  - bounds = Min(p1, p2), Max(p1, p2)

- **Sphere**:  
  - bounds = C-Vector(radius, radius, radius)  
  - C+Vector(radius, radius, radius)
Bounding Boxes

• Triangle:
  • bounds = Min(p1,p2,p3), Max(p1,p2,p3)

• Plane
  • infinite bounds
  • You may want to consider removing planes from your renderer at this point
  • Or keep them outside of your accel structures
Bounding boxes

- **Disc/Ring:**
  \[ C_x \pm rad \sqrt{N_y^2 + N_z^2} \]
  \[ \| \overline{N} \| = 1 \]
  Similar for y/z

- **Group:**
- **Union of object bounding boxes**
- **min of mins**
- **max of maxs**
Uses for bounding boxes

- Quick reject for expensive primitives
- To fill in grid cells
- Directly in Bounding Volume Hierarchy or similar
Bounding Volume Hierarchy

- Observe that intersection is like a search
- We know that searching is $O(n)$ for unsorted lists but $O(\log n)$ for sorted lists
- How do we sort objects from all directions simultaneously?
Bounding volume hierarchy

- Organize objects into a tree
- Group objects in the tree based on spatial relationships
- Each node in the tree contains a bounding box of all the objects below it
At each level of the tree, intersect the ray with the bounding box

- miss: ray misses the entire subtree
- hit: recurse to both children
BVH traversal

- At each level of the tree, intersect the ray with the bounding box
  - miss: ray misses the entire subtree
  - hit: recurse to both children
• At each level of the tree, intersect the ray with the bounding box
  • miss: ray misses the entire subtree
  • hit: recurse to both children
BVH traversal

- At each level of the tree, intersect the ray with the bounding box
  - miss: ray misses the entire subtree
  - hit: recurse to both children
 BVH optimizations

• Stop if the current T value is closer than the BVH node
BVH optimizations

- Stop if the current $T$ value is closer than the BVH node
- Traverse down side of tree that is closer to origin of the ray first
BVH optimizations

- Stop if the current T value is closer than the BVH node
- Traverse down side of tree that is closer to origin of the ray first
- Three or more way split
Building a BVH

- Determining optimal BVH structure is NP-hard problem
- Heuristic approaches:
  - Cost models (minimize volume or surface area)
  - Spatial models
- Categories of approaches:
  - Top down
  - Bottom up
Median cut BVH construction

- Top down approach:
  - Sort objects by position on axis
    - cycle through x,y,z
  - use center of bounding box
  - Insert tree node with half of objects on left and half on right
Weghorst BVH construction

- Bottom up construction
- Add objects one at a time to tree
- Insert to subtree that would cause smallest increase to area
Weghorst BVH construction

- Bottom up construction

- Add objects one at a time to tree

- Insert to subtree that would cause smallest increase to area
Weghorst BVH construction

- Bottom up construction
- Add objects one at a time to tree
- Insert to subtree that would cause smallest increase to area
Weghorst BVH construction

- Bottom up construction
- Add objects one at a time to tree
- Insert to subtree that would cause smallest increase to area
k-d tree

- Recursively divide space in half
- Alternate coordinate axes
- Cycle through axes or store axis split in each node

- What do you do with objects split by the plane?
- Hard part: where do you split in each dimension?
BSP tree

- Like a k-d tree where splitting planes can be arbitrarily located

- Hard part: where do you split in each dimension?

- Harder part: how do you orient each plane?

- More storage/computation, tighter bounds
BSP/k-d tree tradeoffs

- Build is NP-hard problem
- Heuristic approaches are always used
- Traversal is quick
- Storage can be lower than BVH or grid
K-d traversal

- Keep track of intervals on ray:
  - min: 0
  - max: $\infty$

- Compute t of split plane
- If $t > \max$: goto near child
- If $t < \min$: goto far child
- Otherwise: goto both children with updated intervals
K-d traversal

- Keep track of intervals on ray:
  - Compute $t$ of split plane
  - If $t > \text{max}$: goto near child
  - If $t < \text{min}$: goto far child
  - Otherwise: goto both children with updated intervals
K-d traversal

- Keep track of intervals on ray:
  - Compute \( t \) of split plane
  - If \( t > \text{max} \): goto near child
  - If \( t < \text{min} \): goto far child
  - Otherwise: goto both children with updated intervals

Min: 0
Max: \( \infty \)
Split: 1.0
Stack: 1
New min: 0
New max: 1.0
K-d traversal

- Keep track of intervals on ray:
  - Compute $t$ of split plane
  - If $t > \text{max}$: goto near child
  - If $t < \text{min}$: goto far child
  - Otherwise: goto both children with updated intervals
K-d traversal

- Keep track of intervals on ray:
  - Compute t of split plane
  - If t > max: goto near child
  - If t < min: goto far child
  - Otherwise: goto both children with updated intervals

Min: 0
Max: 1.0
Split: 0.4
Stack: 1 5
New min: 0
New max: 0.4
K-d traversal

- Keep track of intervals on ray:
  - Compute t of split plane
  - If t > max: goto near child
  - If t < min: goto far child
  - Otherwise: goto both children with updated intervals

Min: 0.4
Max: 1.0
Split: 0.7
Stack: 1 8
New min: 0.4
New max: 0.6
K-d traversal

- Keep track of intervals on ray:
  - Compute t of split plane
  - If t > max: goto near child
  - If t < min: goto far child
  - Otherwise: goto both children with updated intervals

Min: 0.7
Max: 1.0
Split:
Stack: 1
New min: 0.4
New max: 0.6
K-d traversal

- Keep track of intervals on ray:
  - Compute t of split plane
  - If $t > \text{max}$: goto near child
  - If $t < \text{min}$: goto far child
  - Otherwise: goto both children with updated intervals

Min: 1.0
Max: $\infty$
Split: 1.5
Stack: 10
New min: 1.0
New max: 1.5
K-d traversal

- Keep track of intervals on ray:
  - Compute t of split plane
  - If t > max: goto near child
  - If t < min: goto far child
  - Otherwise: goto both children with updated intervals

Min: 1.0
Max: 1.5
Split: 1.4
Stack: 10 11
New min: 1.0
New max: 1.4
K-d tree build

- Optimal solution is NP-hard
- Spatial median split (a little like octree)
- Object median split (makes balanced trees)
- Cost model based (better)
  - Cost of traversal
  - Cost of intersection
  - Probability of hit
K-d tree build

- Havran ‘01:
- Start with bounding box of scene
- Select split plane along each axis
  - Start with spatial median
  - Move toward bounding box of child nodes
- Recurse
  - Stop if node contains 2-3 objects
  - Stop if depth > max (20-30)
- Backtrack
  - Combine children with identical content
Acceleration Structure Summary

- Most common: Grid, Hierarchical Grid, k-d tree, BVH
- Grid based:
  - P time deterministic build
  - Grid resolution tradeoff
- Tree based:
  - NP hard build
  - Heuristic approaches
Update SimHWRT

- `svn up`
- `make`
- `./update.sh`
  - This one will take a while