The Care and Usage of Vectors

Tom Kelliher, CS 320

Feb. 23, 2005

1 Administrivia

Announcements

Assignment

Carefully read `collision.c` and Section 4.1. Note that `collision.c` is fairly "literate:" good identifier names, high-level overviews, details where necessary, use of whitespace, splitting lines to maximize readability. Use it as a model.

From Last Time

Experimentation with `paint.c`.

Outline

1. Finish-up `paint.c`.
2. Vectors. Data structures and code.
3. Discussion of `collision.c`.

Coming Up

Continued study of `collision.c`.

2 `paint.c`

20 minutes to tie up loose ends. Turn in to `kelliher@goucher.edu` by beginning of next class.

3 Vectors

Operating systems is the application of data structures and algorithms. Computer graphics is the application of data structures, algorithms, trigonometry, linear algebra, Calculus, and differential equations.

1. Scalars.
2. Correspondence between points and vectors.
3. Forming a vector from two points.
4. Vector length: $|u|$.
5. Scaling vectors: αu.
6. Adding vectors.
7. Forming a new point from a point and a vector.
8. Normalizing a vector: $\hat{u} = \frac{1}{|u|}u$.

   Not the same as a normal vector.
9. Dot product properties:

   (a) $|u|^2 = u \cdot u$.

(b) $\cos\theta = \frac{u \cdot v}{|u||v|}$, where θ is the angle between u and v.

(c) The projection of u onto v is $|u|\cos\theta = \frac{u \cdot v}{|v|}$

Note that this is a scalar. What if I need a vector?

3.1 Data Structures and Code

This is all 2-D. It should be extended to 3-D.

1. Basic data structure:

```
typedef struct Vector2
{
   GLdouble x, y;
} Vector2;
```

2. Vector length:

```
double distanceSquared(double x, double y)
{
   return x * x + y * y;
}

double vectorLength(Vector2 v)
{
   return sqrt(distanceSquared(v.x, v.y));
}
```

   Square root and division are expensive. Avoid where possible.

3. Scalar, vector product:

```
Vector2 scalarProduct(double s, Vector2 v)
{
   v.x *= s;
   v.y *= s;

   return v;
}
```

4. Dot product:

```
double dotProduct(Vector2 a, Vector2 b)
{
   return a.x * b.x + a.y * b.y;
}
```

5. Vector normalization:

```
Vector2 normalize(Vector2 v)
{
   double length = vectorLength(v);

   v.x /= length;
   v.y /= length;

   return v;
}
```

6. Vector sum:

```
Vector2 vectorSum(Vector2 a, Vector2 b)
{
   a.x += b.x;
   a.y += b.y;

   return a;
}
```

4 collision.c

1. `main()`: Note use of `srand()`.

2. `init()`: Calls `initBalls()`, little else.

3. `initBalls`:

   (a) First off, we need a better object manager. More precisely, we need an object manager.

(b) The "object" definitions:

```
typedef struct Color
{
   GLdouble r, g, b;
} Color;

typedef struct Ball
{
   Vector2 position; /* Ok, so it's not really a vector. Sue me. */
   Vector2 velocity;
   GLdouble radius;
   GLdouble mass;
   Color color;
   GLuint handle;
} Ball;
```

These should be proper classes.

(c) Note setting of basic object attributes. These should be kept in a file and handled by the object manager.

(d) Note compilation of object rendering primitives via individual display lists.

4. `placeBalls()`:

(a) Placement of the first ball along unit circle.

Velocity computation. Target: origin. Scaling velocity.

Translating position to circle of radius 40.

(b) Avoiding initial collision: Constrained placement of second ball; $\pi/4$ or more away.

(c) Options: Aiming second ball at a point other than the origin. Computing and normalizing the velocity vector.

Making the second ball stationary, at an arbitrary position.