Catmull-Rom Interpolator

```
void CatmullRomInterp::RecalculateVectors(void){

	// ***--- Switch for choosing what type for first point ---***
	switch (begin)
	{
	case STOP:
		keys[0].vIn = vector(0,0,0);
		keys[0].vOut = vector(0,0,0);
		break;
	case LINEAR:
		keys[0].vIn = keys[1].p - keys[0].p;
		keys[0].vOut = keys[0].vIn;
		break;
	case QUADRATIC:
		keys[0].vIn = .5 * (3 * ( keys[1].p -  keys[0].p ) - ( keys[2].p -
			keys[1].p ) );
		keys[0].vOut = keys[0].vIn;
		break;
	default:
		complain("Invalid beginning condition for catmul rom: %d", begin);
	};

	// ***--- CatmullRom Calculation for all inbetween points ---***
	for (int i = 1; i < nKeys - 1; i++){
		vector T0 = keys[i].p - keys[i-1].p;
		vector T1 = keys[i+1].p - keys[i].p;

		keys[i].vIn = ( T1.length() / ( T0.length() + T1.length() ) ) * T0 + (
			T0.length() / ( T0.length() + T1.length() ) ) * T1;
		keys[i].vOut = keys[i].vIn;
	}

	// ***--- Switch for choosing what type for end point ---***
	switch (end)
	{
	case STOP:
		keys[NKeys()-1].vIn = vector(0,0,0);
		keys[NKeys()-1].vOut = vector(0,0,0);
		break;
	case LINEAR:
		keys[NKeys()-1].vIn = keys[NKeys()-1].p - keys[NKeys()-2].p;
		keys[NKeys()-1].vOut = keys[NKeys()-1].vIn;
		break;
	case QUADRATIC:
		keys[NKeys()-1].vIn = .5 * (3 * ( keys[NKeys()-1].p -  keys[NKeys()-2].p )
			- ( keys[NKeys()-2].p -  keys[NKeys()-3].p ) );
		keys[NKeys()-1].vOut = keys[NKeys()-1].vIn;
		break;
	default:
		complain("Invalid beginning condition for catmul rom: %d", begin);
	};
	RebuildParamDistanceTable();
}
```

Catmull-Rom Interpolator

```
void Interpolator::RebuildParamDistanceTable(void){
	if (nKeys == 0)
		return;

	if (distTable != NULL)
		delete [] distTable;

	distTable = new ParamDistEntry[100 * (nKeys - 1) + 1];

	double totalLen = 0;
	double dS = .01;
	point P0, P1;

		for (int i = 0; i < nKeys; i++){
		//special case for the first key
		if(i == 0){
			distTable[0].d = 0;
			distTable[0].s = 0;
		}else{
			for (int j = 1; j < 101; j++){
				// calculate paramater for the table
				distTable[((i-1)*100)+j].s = (i-1) + (j*dS);

				//calculate the difference in time
				P0 = pointOnCurve((j-1) * dS);
				P1 = pointOnCurve(j * dS);
				//add the distance to the total length
				totalLen += dist(P0,P1);

				//calculate distance for the table
				distTable[((i-1)*100)+j].d = totalLen;

				/*cout << distTable[((i-1)*100)+j].s;
				cout << "     ";
				cout << distTable[((i-1)*100)+j].d;
				cout << "\n";*/
			}
		}
		keys[i].dist = distTable[i * 100].d;
	}
	//averageDT = totalLen / nKeys;
	averageDT = 0;
	for (int i = 0; i < nKeys - 1; i++){
		averageDT += (keys[i + 1].t - keys[i].t);
	}
	// Calculate the average interval length
	averageDT /= nKeys - 1;
}


point Interpolator::Evaluate(double t)  {
	int i = FindInterval(t);

	if(nKeys < 2){

		return pointOnCurve(0);
```

Catmull-Rom Interpolator

```
	}else if (!keys[i].constant){

		// Calculate s which is the % along and add to it i which is the interval
		double s = ((t-keys[i].t) / ( keys[i+1].t - keys[i].t )) + i;

		return pointOnCurve(s);
	}
	else
	{

		double s = 0;

		double speed = (keys[i+1].dist - keys[i].dist) /( keys[i+1].t - keys[i].t);
		double distance = keys[i].dist + (t - keys[i].t) * speed;

		int found = 0;

		while (distTable[found].d < distance){
			found++;
		}

		s = distTable[found].s + ( ( (distance - distTable[found].d) /
			(distTable[found+1].d - distTable[found].d) ) *
			(distTable[found+1].s - distTable[found].s) );


		return (pointOnCurve(s));
	}
}
```