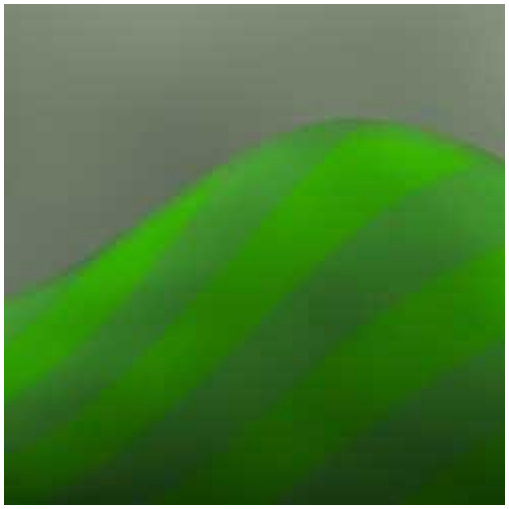
如何制作一个类似 tiny wings 的游戏：第一部分

原文链接地址:http://www.raywenderlich.com/3888/how-to-create-a-game-like-tiny-wings-part-1

PS:这两篇文章已经由游戏邦翻译了，不过排版格式可能不利于大家实践，代码使用的全部是图片。本来我是不准备重复发明轮子的，但是，想想，如果能帮大家节省几分钟的时间，我觉得重复发明轮子也是值得的。：）另外，还有一个好消息，知易出新的 box2d 教程了，我们学习 BOX2D 的朋友有更多的资源可以学习了。：）

教程截图：



Tiny Wings 是一款非常流行的游戏，由 Andreas Illiger 开发制作，里面有一只小鸟，试图追赶上飞过山头的旅程。

乍一看，Tiny Wings 的游戏玩法非常简单，但是，也有许多地方很奇妙。这些山丘和纹理是动态创建的，同时游戏使用 box2d 物理引擎来仿真小鸟的移动。

由于这个游戏太火了，而且有一些比较酷的技术要点，许多开发者对此非常感兴趣。

这个教程是基于一个非常好的 demo project 来写成的，作者是 Sergey Tikhonov。他实现了 Tiny Wings 的一些很酷的特性，感谢 Sergey!

这个教程系列被分为以下 3 个部分：

- *基础:* 首先阅读如何使用 CCRenderTexture 来创建动态纹理教程，这个教程向你展示了如何动态创建山丘和背景纹理，这些技术将在本教程的后面使用到。
- *第一部分:* 本教程就是第一部分！这个部分将向你展示如何动态生成 Tiny Wings 里面的小山丘。
- *第二部分:* 它将向你展示如何为 Tiny Wings 这样的游戏添加 box2d 游戏玩法。

这个教程假设你对 cocos2d 已经非常熟悉了。如果你对 cocos2d 还很陌生的话，请先阅读本博客翻译的其它 cocos2d 教程。

Getting Started

如果你还没有上一个教程的工程，那么可以在这里下载样例工程。

接下来，为 terrain 创建一个类，点击 File\New\New File，再选择 iOS\Cocoa Touch\Objective-C class，然后点击 Next。使得它是 CCNode 的一个子类，然后点击 Next，并把类命名为 Terrain.m，最后点击 Save.

然后打开 Terrain.h，并用以下内容替换之：

```
#import "cocos2d.h"
```

```
@class HelloWorldLayer;

#define kMaxHillKeyPoints 1000

@interface Terrain : CCNode {
int _offsetX;
CGPoint _hillKeyPoints[kMaxHillKeyPoints];
CCSprite *_stripes;
}

@property (retain) CCSprite * stripes;
- (void) setOffsetX:(float)newOffsetX;

@end
```

这里声明了一个数组，叫做_hillKeyPoints，它里面存储了每一个山丘峰值的顶点，同时定义了当前被滚动的地形的偏移量。

接下来，我们将实现 Terrain.m。我将一步步来解释给你听，继续并把 Terrain.m 里面的内容全部删除，然后用下面的代码替换之：

```
#import "Terrain.h"
#import "HelloWorldLayer.h"

@implementation Terrain
@synthesize stripes = _stripes;

- (void) generateHills {

CGSize winSize = [CCDirector sharedDirector].winSize;
float x = 0;
float y = winSize.width / 2;
for(int i = 0; i < kMaxHillKeyPoints; ++i) {
_hillKeyPoints[i] = CGPointMake(x, y);
x += winSize.width/2;
y = random() % (int) winSize.height;
```

```
}

}
```

这个方法用来随机生成每个山丘的顶点。这是一个非常简单的实现，仅仅是让我们从这里开始。

第一个点在屏幕的左边界，在 y 轴的中间。它之后的每一个点都是在 x 方向偏移半个屏幕宽度，y 方向就是一个随机产生的值，范围从 0 到屏幕的高度。这样就会产生高低不同的山丘。.

```
- (id)init {
if ((self = [super init])) {
[self generateHills];
}
return self;
}

- (void) draw {

for(int i = 1; i < kMaxHillKeyPoints; ++i) {
ccDrawLine(_hillKeyPoints[i-1], _hillKeyPoints[i]);
}

}
```

init 方法调用 generateHills 来创建山丘，然后 draw 方法就是简单的把每一个山头的顶点用线连接起来，目的是为了调试。这样的话，我们就可以很容易地在屏幕上可视化。

```
- (void) setOffsetX:(float)newOffsetX {
_offsetX = newOffsetX;
self.position = CGPointMake(-_offsetX*self.scale, 0);
}

- (void)dealloc {
```

```
[_stripes release];
_stripes = NULL;
[super dealloc];
}

@end
```

考虑一下地形如何移动---当我们的主角沿着地形的 x 轴方向运动的时候，地形本身要往左边滑动。因此，我们不得不把偏移量乘以-1，同时不要忘了把 scale 考虑进去！

是时候测试它们了。回到 HelloWorldLayer.h，然后作如下修改：

```
// Add to top of file
#import "Terrain.h"

// Add inside @interface
Terrain * _terrain;
```

然后打开 HelloWorldLayer.m 并作如下修改：

```
// Add inside init BEFORE call to genBackground
_terrain = [Terrain node];
[self addChild:_terrain z:1];

// Add at bottom of update
[_terrain setOffsetX:offset];

// Modify genBackground to the following
- (void)genBackground {

[_background removeFromParentAndCleanup:YES];

ccColor4F bgColor = [self randomBrightColor];
_background = [self spriteWithColor:bgColor textureSize:512];
```

```
CGSize winSize = [CCDirector sharedDirector].winSize;
_background.position = ccp(winSize.width/2, winSize.height/2);
ccTexParams tp = {GL_LINEAR, GL_LINEAR, GL_REPEAT, GL_REPEAT};
[_background.texture setTexParameters:&tp];

[self addChild:_background];

ccColor4F color3 = [self randomBrightColor];
ccColor4F color4 = [self randomBrightColor];
CCSprite *stripes = [self stripedSpriteWithColor1:color3 color2:color
4 textureSize:512 stripes:4];
ccTexParams tp2 = {GL_LINEAR, GL_LINEAR, GL_REPEAT, GL_CLAMP_TO_EDGE};
[stripes.texture setTexParameters:&tp2];
_terrain.stripes = stripes;

}
```

注意，这里每一次点击屏幕都会随机生成一个带状的纹理地形，这样做是为了方便测试。

同时，当调用在 update 方法里面调用_background 的 setTextureRect 方法时，你可能希望把 offset 乘以 0.7，这样可以使背景比地形稍微移动慢一些。

就这么多！编译并运行代码，现在你可以看到一条线，它连接了每一个山丘的顶点。

当你看着你的山丘滚动的时候，你可能很快会意识到，这样子并不会工作地很好。由于我们采用随机决定 y 轴的值，有时候，一些山丘太大，而有时候一些山丘则太小。同时，x 轴方向很没什么变化，这跟 Tiny Wings 游戏比起来差远了。

但是，现在，你有了这样一份测试代码可以跑起来了，而且有一种比较好的方法来可视化并且支持调试。是时候想一种更好的算法了！

你要么可以想一会会儿，然后提出你自己的生成山丘的算法，然后替换掉 generateHills 方法。或者，你可以使用 Sergey 的实现，我们会在下个部分向你展示！

一种更好的生成山丘的算法

如果你选择 Sergey 的实现，那么把 Terrain.m 里面的 generateHills 方法替换成下面的样子：

```
- (void) generateHills {

CGSize winSize = [CCDirector sharedDirector].winSize;

float minDX = 160;
float minDY = 60;
```

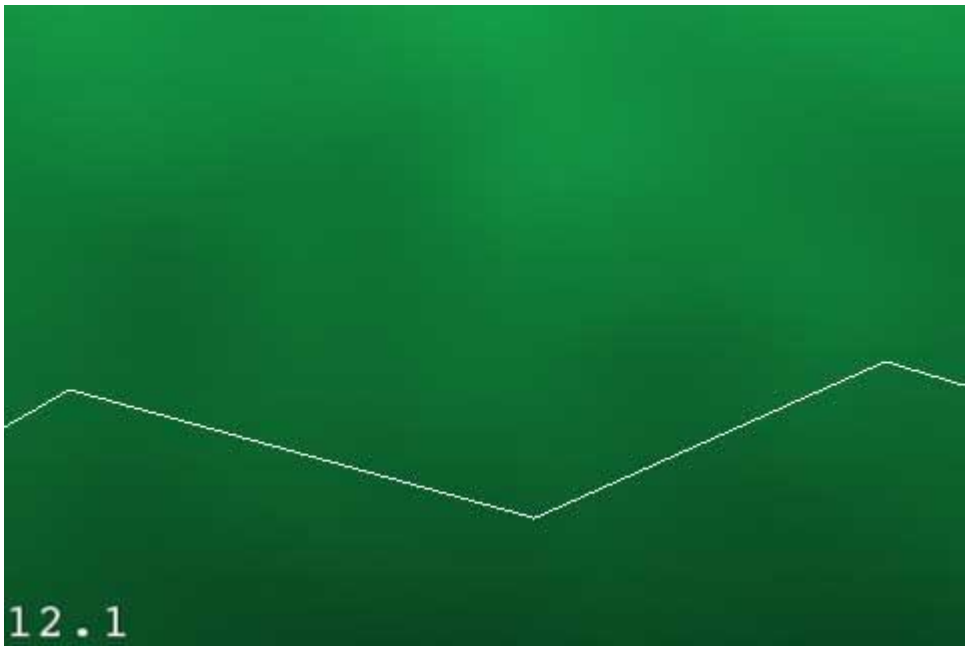
```
int rangeDX = 80;
int rangeDY = 40;

float x = -minDX;
float y = winSize.height/2-minDY;

float dy, ny;
float sign = 1; // +1 - going up, -1 - going down
float paddingTop = 20;
float paddingBottom = 20;

for (int i=0; i<kMaxHillKeyPoints; i++) {
_hillKeyPoints[i] = CGPointMake(x, y);
if (i == 0) {
x = 0;
y = winSize.height/2;
} else {
x += rand()%rangeDX+minDX;
while(true) {
dy = rand()%rangeDY+minDY;
ny = y + dy*sign;
if(ny < winSize.height-paddingTop && ny > paddingBottom) {
break;
}
}
y = ny;
}
sign *= -1;
}
}
```

这个算法的策略如下所示：

- 递增 X 轴，范围是 160 加上一个 0-80 的随机数
- 递增 Y 轴，范围是 60 加上一个 0-40 的随机数
- 异常：每隔一个顶点逆转一次 Y 轴坐标

- 不要让 Y 轴的值太接近屏幕的顶部或底部了(paddingTop, paddingBottom)
- 刚开始时，硬编码第一个点的坐标是 (0, winSize.height/2)，这样的话，我们的山丘就是从屏幕左边出现的。

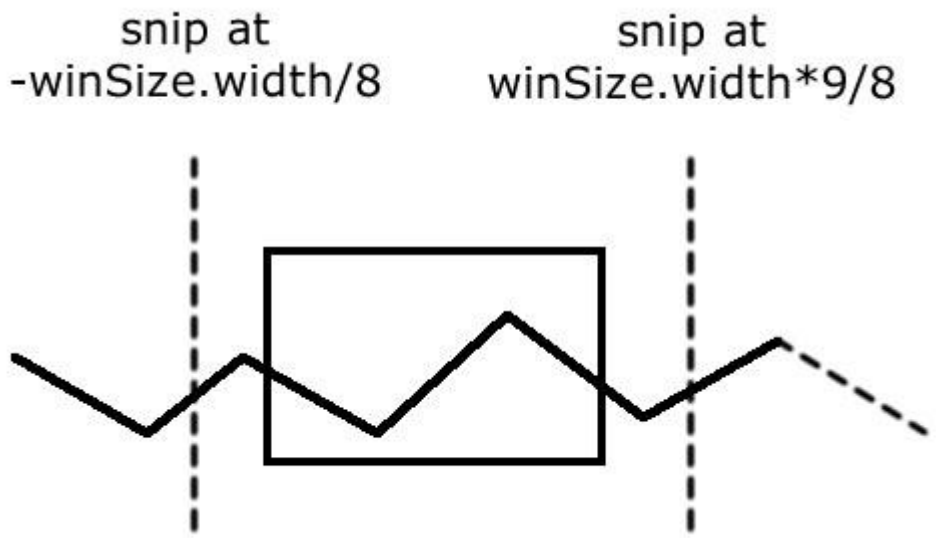
编译并运行代码，现在，你将看到一个更好的山丘算法了，你可以想像一下，有一只海豹在上面飞过去的样子！：）



一次画一个部分

在我们继续之前，我们需要做一次很重要的性能优化。目前为止，我们画了山丘的 1000 个峰值点，虽然它们中只有一小部分显示在屏幕上面。

因此，我们只需要根据显示的屏幕区域范围大小来计算哪些峰值点将会被显示出来，然后我们就只显示那些点，如下所示：



让我们把这个实现一下。打开 Terrain.h，然后添加两个实例变量：

```
int _fromKeyPointI;
int _toKeyPointI;
```

然后打开 Terrain.h,并在 init 方法的上面添加一个新的方法，叫做 resetHillVertices。具体如下所示：

```
- (void)resetHillVertices {

CGSize winSize = [CCDirector sharedDirector].winSize;

static int prevFromKeyPointI = -1;
static int prevToKeyPointI = -1;

// key points interval for drawing
while (_hillKeyPoints[_fromKeyPointI+1].x < _offsetX-winSize.width/8/
self.scale) {
_fromKeyPointI++;
}
while (_hillKeyPoints[_toKeyPointI].x < _offsetX+winSize.width*9/8/se
lf.scale) {
_toKeyPointI++;
}
```

```
}
```

这里，我们遍历所有的峰值点（从 0 开始），然后比较 x 轴。

不管当前设置给屏幕左边界的偏移量是多少，我们只需要把它减掉 winSize.width/8。如果小于上面的差值，那么就继续递增，直到找到一个大的峰值点。同样的，我们可以找到 toKeyPoint。

现在，让我们检测一下，看这个能跑起来不！修改你的 draw 方法，如下所示：

```
- (void) draw {

for(int i = MAX(_fromKeyPointI, 1); i <= _toKeyPointI; ++i) {
glColor4f(1.0, 0, 0, 1.0);
ccDrawLine(_hillKeyPoints[i-1], _hillKeyPoints[i]);
}

}
```

现在，我们不是绘制所有的点，而只是绘制屏幕可见范围之内的点。同时，我们把线的颜色改成红色，这样可以更容易地分辨出来。

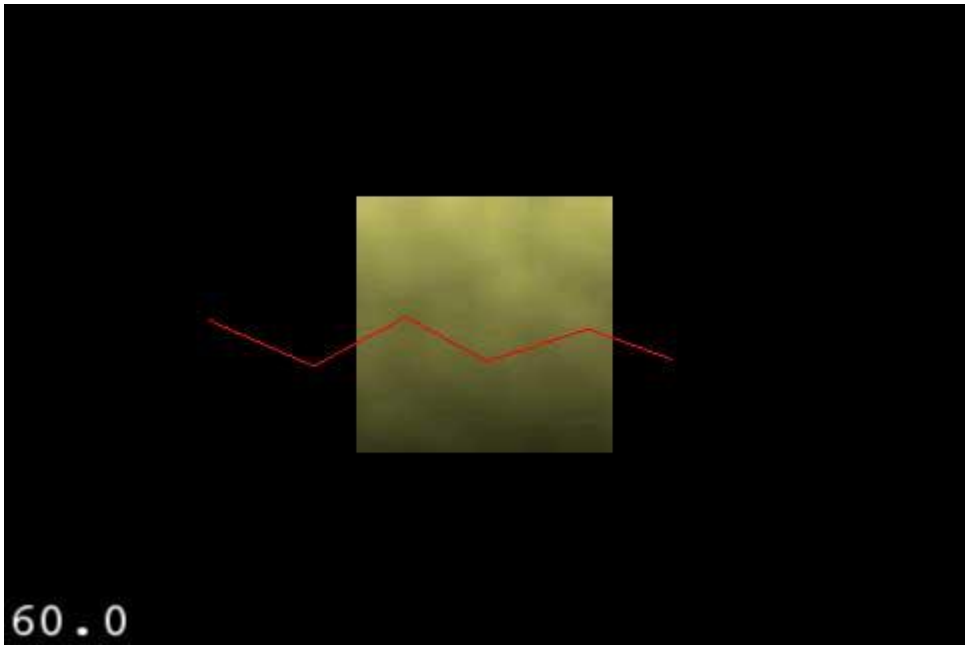
接下来，往 Terrain.m 中做更多一些的修改，如下所示：

```
// Add at bottom of init
[self resetHillVertices];

// Add at bottom of setOffsetX
[self resetHillVertices];
```

还有一件事---为了使它能够被看见，我们打开 HelloWorldLayer.mm 文件，并在 init 方法中添加下面的语句：

```
self.scale = 0.25;
```

编译并运行你的代码，现在，你应该能够看到，线段只是当它们该出现的时候才被绘制了。

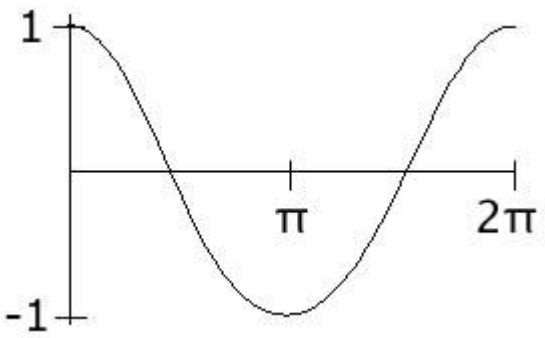


制作光滑的斜坡

目前为止还不错，但是，我们有一个很大我问题--这一点也不像山丘啊！在现实生活中，山丘可不是直上直下的直线啊！他们有坡度。

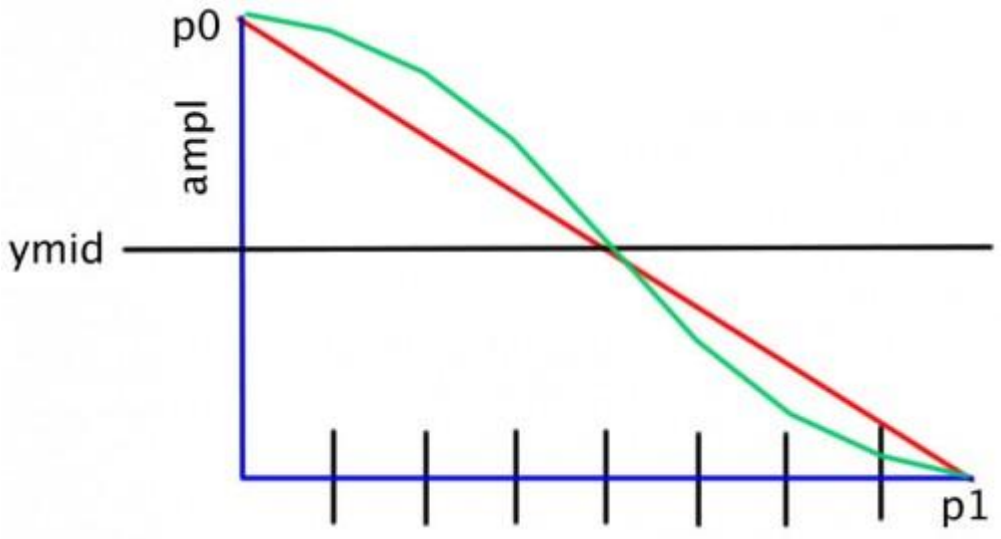
但是，我们怎么让我们的山丘变成曲线呢？其实我们高中时候就学过了，就是 cosine 函数！

作为知识的回顾，下面给出 cosine 曲线的图示：



因此，它是从 1 开始，然后每隔 PI 个单位长度弯曲到-1.

但是，我们怎么充分利用这个函数来创建一个非常好的曲线来连接我们的 keypoints 呢？让我们先考虑两个点的情况，如下图所示：



首先，我们需要分段来画线。因此，我们将每 10 个像素点画一个线段。类似的，我们想到一个完整的 cosine 曲线，因此，我们可以用 PI 除以线段的个数来得到每一个顶点的 delta 角度。

然后，我们让 cos(0)为 p0 的 Y 坐标，cos(PI)为 P1 的 Y 坐标。因此，我们将调用 cos(angle)，同时用 p1 和 p0 间的距离乘以 cos(angle）。（这个距离如上图 ampl 所示）

因为 cos(0)=1，而且 cos(PI)=-1，所以，ampl 在 p0 位置，-ampl 在 p1 位置。我们可以把它加上中点位置坐标值，这样就可以得到我们想到的 Y 坐标值！

让我们看看具体代码是什么样子。首先，在 Terrain.h 的顶部定义每一个线段的长度：

```
#define kHillSegmentWidth 10
```

然后往 draw 方法中添加如下代码，添加位置紧随 ccDrawLine 调用之后：

```
glColor4f(1.0, 1.0, 1.0, 1.0);

CGPoint p0 = _hillKeyPoints[i-1];
CGPoint p1 = _hillKeyPoints[i];
int hSegments = floorf((p1.x-p0.x)/kHillSegmentWidth);
```

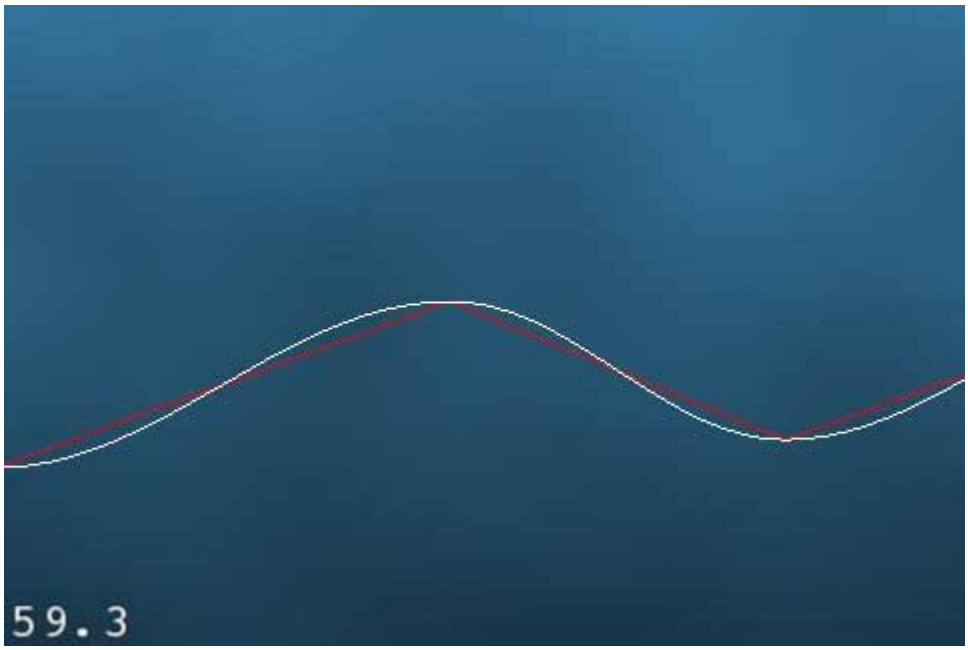
```
float dx = (p1.x - p0.x) / hSegments;
float da = M_PI / hSegments;
float ymid = (p0.y + p1.y) / 2;
float ampl = (p0.y - p1.y) / 2;

CGPoint pt0, pt1;
pt0 = p0;
for (int j = 0; j < hSegments+1; ++j) {

pt1.x = p0.x + j*dx;
pt1.y = ymid + ampl * cosf(da*j);

ccDrawLine(pt0, pt1);

pt0 = pt1;

}
```

这里就完全实现了我们之前讨论的策略。花上一分钟时间仔细想一想这段代码，确保你完全理解它是如何工作的，因为，接下来我们将基于这里所做的工作。

最后一件事---我们不再需要缩小了，因此，在 HelloWorldLayer.mm 中，把 init 方法中的 scale 的设置改成 1.0：

```
self.scale = 1.0;
```

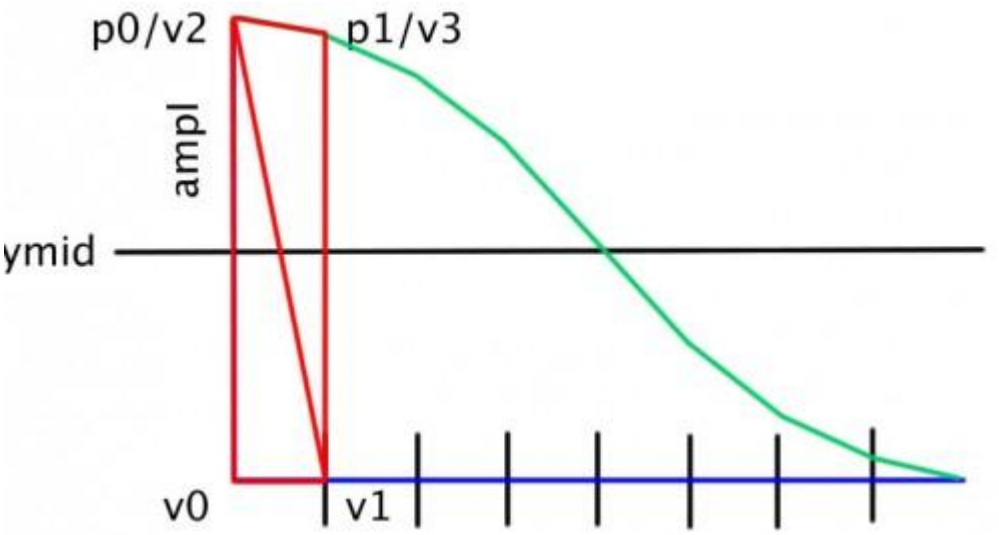
编译并运行，现在你可以看到有一条曲线连接起山丘了！



绘制山丘

现在，我们知道如何用曲线来代表这些山丘的顶部了，现在我们用上个教程中学会的带状纹理的制作方法，可以非常简单地用带状纹理来绘制这些山丘了！

计划是，对于山丘的每一段，我们将计算出两个三角形来渲染山丘，如下图所示：



我们也将设置每一个点的纹理坐标。对于 X 轴来说，我们简单地除以纹理的宽度（因为纹理是重复的）。对于 Y 轴，我们把山丘的底部映射成 0，顶部映射成 1，并沿着条带的方向分配纹理高度。

为了实现这个，首先，让我们在 Terrain.h 中做如下更改：

```
// Add some new defines up top
#define kMaxHillVertices 4000
#define kMaxBorderVertices 800

// Add some new instance variables inside the @interface
int _nHillVertices;
CGPoint _hillVertices[kMaxHillVertices];
CGPoint _hillTexCoords[kMaxHillVertices];
int _nBorderVertices;
CGPoint _borderVertices[kMaxBorderVertices];
```

然后，在 Terrain.m 的 resetHillVertices 方法的底部添加下列代码：

```
if (prevFromKeyPointI != _fromKeyPointI || prevToKeyPointI != _toKeyP
ointI) {

// vertices for visible area
_nHillVertices = 0;
_nBorderVertices = 0;
CGPoint p0, p1, pt0, pt1;
p0 = _hillKeyPoints[_fromKeyPointI];
for (int i=_fromKeyPointI+1; i<_toKeyPointI+1; i++) {
p1 = _hillKeyPoints[i];

// triangle strip between p0 and p1
int hSegments = floorf((p1.x-p0.x)/kHillSegmentWidth);
float dx = (p1.x - p0.x) / hSegments;
float da = M_PI / hSegments;
float ymid = (p0.y + p1.y) / 2;
float ampl = (p0.y - p1.y) / 2;
pt0 = p0;
_borderVertices[_nBorderVertices++] = pt0;
for (int j=1; j<hSegments+1; j++) {
pt1.x = p0.x + j*dx;
```

```
pt1.y = ymid + ampl * cosf(da*j);
_borderVertices[_nBorderVertices++] = pt1;

_hillVertices[_nHillVertices] = CGPointMake(pt0.x, 0);
_hillTexCoords[_nHillVertices++] = CGPointMake(pt0.x/512, 1.0f);
_hillVertices[_nHillVertices] = CGPointMake(pt1.x, 0);
_hillTexCoords[_nHillVertices++] = CGPointMake(pt1.x/512, 1.0f);

_hillVertices[_nHillVertices] = CGPointMake(pt0.x, pt0.y);
_hillTexCoords[_nHillVertices++] = CGPointMake(pt0.x/512, 0);
_hillVertices[_nHillVertices] = CGPointMake(pt1.x, pt1.y);
_hillTexCoords[_nHillVertices++] = CGPointMake(pt1.x/512, 0);

pt0 = pt1;
}

p0 = p1;
}

prevFromKeyPointI = _fromKeyPointI;
prevToKeyPointI = _toKeyPointI;
}
```

这里的大部分代码看起来都很熟悉，因为，我们在前面一部分中绘制 cosine 曲线的时候已经讲解过了。

新的部分就是，我们将用山丘的每一个线段的顶点来填充一个顶点数组，和上面策略部分讲解过的一样。每一条带子需要 4 个顶点和 4 个纹理坐标点。

现在，你可以在 draw 方法的顶部添加下面代码了：

```
glBindTexture(GL_TEXTURE_2D, _stripes.texture.name);
glDisableClientState(GL_COLOR_ARRAY);

glColor4f(1, 1, 1, 1);
glVertexPointer(2, GL_FLOAT, 0, _hillVertices);
```

```
glTexCoordPointer(2, GL_FLOAT, 0, _hillTexCoords);
glDrawArrays(GL_TRIANGLE_STRIP, 0, (GLsizei)_nHillVertices);
```

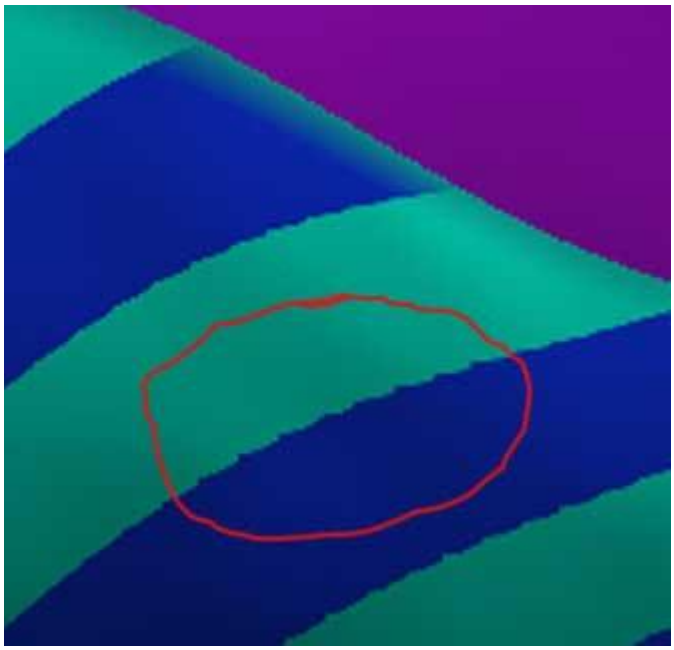
这里绑定带状纹理作为 opengl 的渲染纹理来使用，然后传入之前计算好的顶点数组和纹理坐标数组，然后以三角带模式来绘制这些组织中的顶点。

同时，你可以注释掉画山丘峰值线和曲线的代码，因为你想让它看起来效果更好，而且并不想调试了。

编译并运行代码，现在你可以看到下面非常酷的山丘效果了！

不完美?

如果你仔细看看这些山丘，你可能注意到了一些不完美的地方，如下所示：

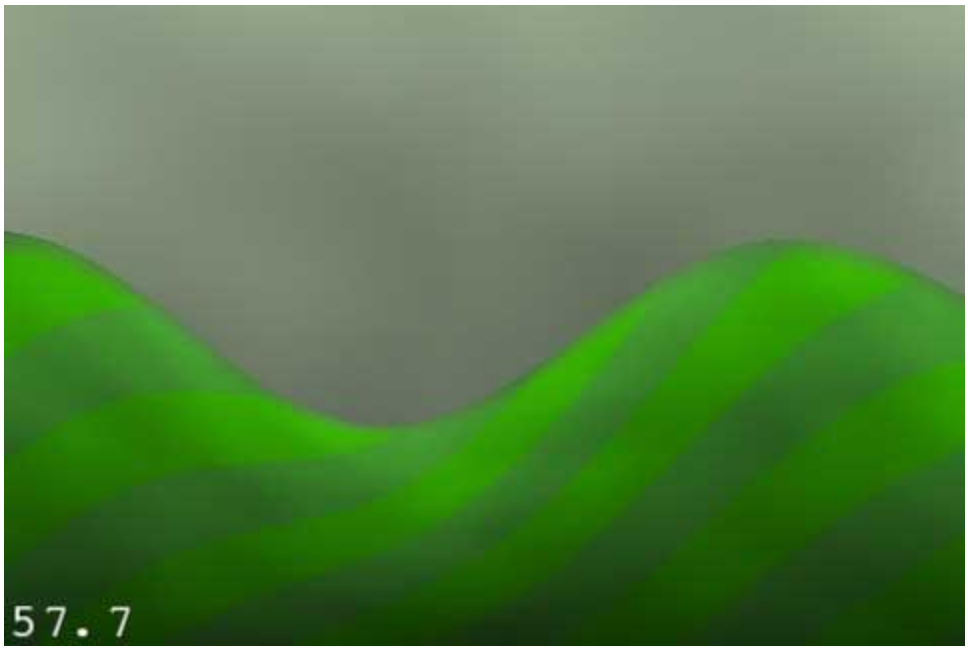


在 Cocos2D forums 里面有一些人指出，你可以增加更多的顶点线段来解决这个问题。（我们上面目前是使用一个分段（segment））。

然后，我个人发现，增加垂直顶点线段的个数并没有提高多少质量，但是，增加水平方向的线段个数可以办到。开打 Terrain.h，然后修改 kHillSegmentWidth，如下所示：

```
#define kHillSegmentWidth 5
```

再一次运行，你会看到这次生成的山丘更好了！



何去何从?

这里有本教程的完整源代码。

接下来，请期待本教程的第二部分。

PS：本人水平有限，翻译不准的地方，望大家一定要指出来啊！