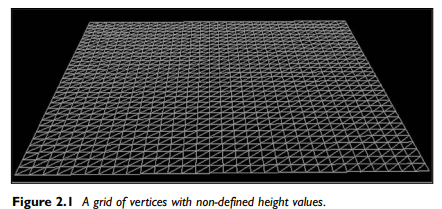
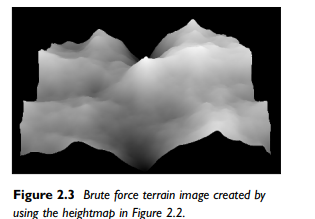
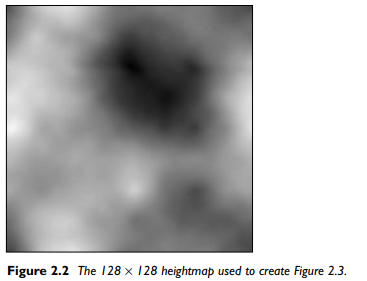
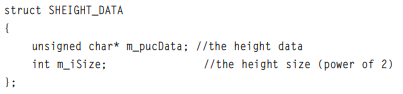
1. Chương 1: Height map
   1. Nội dung
      * Hightmap là gì, làm thế nào để tạo chúng, load chúng như thế nào.
      * Làm thế nào để kiết xuất địa hình sử dụng thuật toán brute force.
      * Làm thế nào để generate fractal địa hình sử dụng hai thuật toán: fault formation và midpoint displacement.
   2. Heightmaps
      * Ta có lưới đa giác như sau được tạo thành từ các đa giác(dọc theo phương X và Z) thông thường và không có giá trị chiều cao:



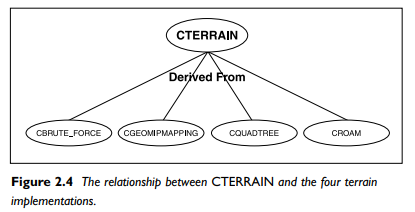
* + - Heightmap là một danh sách (series) các biến có kiểu unsigned char(giá trị từ 0-255) màu xám để gán cho từng điểm. Màu càng tối thì độ cao càng thấp hơn càng sáng thì độ cao càng cao hơn.



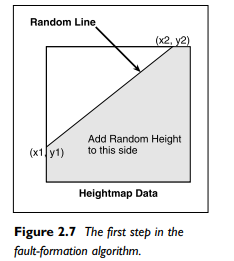
* + - Thông thường định dạng của định dạng heightmap có kiểu định dạng raw. Vì nó chứa dữ liệu thuần túy đơn giản dễ dàng hơn để tải sơ đồ chiều cao vào.
    - Cấu trúc đơn giản của một đối tượng heightmap như sau



* 1. Tạo class BaseTerrain
     + Chúng ta cần tạo một lớp BaseClass mà từ đó có tất cả các địa hinh của thể của chúng ta: brute force, geomipmapping, v.v. Đây là hình ảnh của ý tưởng đó.

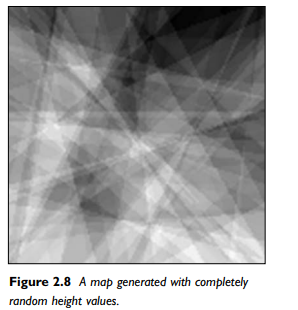


* 1. Fractal Terrain Generation
     + Có 2 thuật toán tạo địa hình Fractal terrain generation: fault formation và midpoint displacement.
     + Thuật toán Fault formation
       - Một phương pháp tạo địa hình fractal được gọi là sự hình thành đứt gãy(fault formation). Fault formation là quá trình tạo ra các “đứt gãy(faults)” trên địa hình; cho phần lớn, nó tạo ra địa hình khá bằng phẳng. Về cơ bản, tất cả những gì chúng tôi làm là thêm một dòng(line) ngẫu nhiên vào trường chiều cao trống, sau đó chúng tôi thêm một dòng ngẫu nhiên chiều cao sang một bên. Xem Hình 2.7 nếu bạn gặp sự cố hình dung điều này hoặc nếu bạn chỉ muốn xác nhận rằng hình ảnh trong đầu của bạn (hoặc, nếu bạn giống tôi, giọng nói trong đầu bạn—tôi lạ như vậy) là đúng.

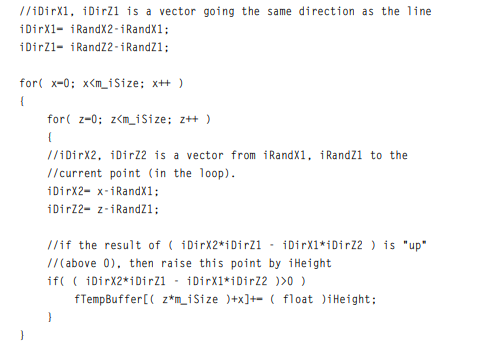


* + - * Tất nhiên, đó chỉ là bước đầu tiên trong toàn bộ quá trình. vẫn còn một số điều bạn cần biết về thuật toán trước khi tiếp cận các giai đoạn nâng cao của nó. Đầu tiên, giá trị chiều cao mà tôi đã nói đến trước đó cần phải được giảm với mỗi lần lặp lại. Tại sao, bạn có thể hỏi? Chà, nếu bạn không giảm chiều cao sau mỗi lần vượt qua, bạn sẽ kết thúc lên bằng sơ đồ chiều cao như Hình 2.8. Xem Hình 2.9 để biết ví dụ về bản đồ chiều cao sẽ trông như thế nào.
      * Lưu ý, trong Hình 2.8, các đốm sáng/tối không có vần điệu hoặc lý do như thế nào; họ chỉ trải ra khắp nơi. Điều này sẽ tốt cho địa hình hỗn loạn, nhưng chúng tôi muốn tạo ra những ngọn đồi thoai thoải. Không có nỗi sợ; sửa chữa vấn đề này là khá đơn giản. Chúng tôi muốn giảm tuyến tính giá trị chiều cao mà không làm nó giảm xuống 0. Để làm điều này, chúng tôi sử dụng các phương trình sau (lấy từ demo2\_2):

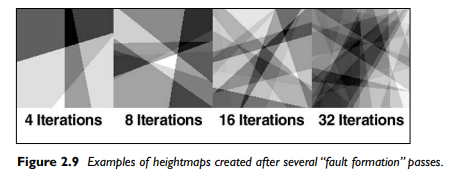




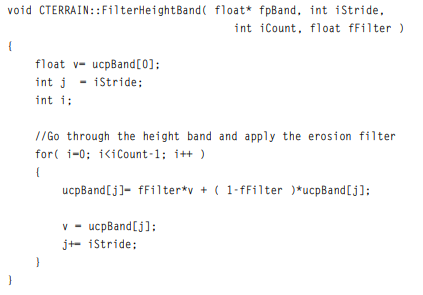
* + - * iMinDelta, iMaxDelta và iIterations đều được cung cấp dưới dạng function arguments. iMinDelta và iMaxDelta đại diện cho mức thấp nhất và cao nhất (tương ứng) các giá trị mà bạn muốn cho chiều cao khi tạo mới lỗi lầm. Tôi có xu hướng gắn bó với giá trị 0 cho iMinDelta và giá trị 255 cho iMaxDelta. iIterations, như tôi đã nói trước đây, đại diện cho số lượng lỗi vượt qua để thực hiện (số lần khác nhau mà sơ đồ chiều cao phải được chia). Và cuối cùng nhưng không kém phần quan trọng, iCurrentIteration đại diện cho số lần lặp hiện tại
      * Như tôi đã nói trước đó, chúng tôi chỉ muốn nâng cao một bên của ranh giới, và chúng tôi muốn tăng giá trị chiều cao của mọi điểm ở phía đó của đường thẳng. Do đó, chúng ta sẽ phải lặp qua tất cả các giá trị chiều cao cho toàn bộ sơ đồ chiều cao. Tất cả điều này là dễ dàng để thực hiện; chỉ là liên quan đến một số toán học đơn giản. Chúng tôi có một vectơ đi theo hướng của đường thẳng của chúng tôi (được xác định bởi hai điểm ngẫu nhiên mà chúng tôi được tạo trước đó) và hướng của nó được lưu trữ trong (iDirX1, iDirZ1). Các vectơ tiếp theo mà chúng tôi muốn tạo là một vectơ từ ngẫu nhiên ban đầu điểm (iRandX1, iRandZ1) ​​đến điểm hiện tại trong vòng lặp (x, z). Sau đó thế là xong, chúng ta cần tìm thành phần Z của tích chéo, và nếu nó lớn hơn 0, thì chúng ta cần nâng dòng điện điểm trong câu hỏi. Tất cả các giải thích trước đó được hiển thị tiếp theo trong mã từ bản demo.



* + - * Trong khi bạn đang nhìn vào fault formation and midpoint displacement code trong các hai phân đoạn và trong demo2\_2, bạn có thể lưu ý cách tôi tạo bộ đệm tạm thời, fTempBuffer, của các giá trị dấu phẩy động để dính tất cả các giá trị chiều cao trong. Nếu bạn còn nhớ, mặc dù vậy, tôi đã nói về bản đồ chiều cao của chúng tôi là một mảng biến unsigned char.Tại sao tôi sẽ sử dụng các biến dấu phẩy động trong này tình huống? Tôi đã làm điều này bởi vì các thuật toán cần một mức độ chính xác cao hơn so với của chúng tôi bộ đệm chiều cao char không dấu bình thường. Sau khi chúng tôi đã tạo và chuẩn hóa toàn bộ sơ đồ chiều cao, tôi chuyển tất cả thông tin từ fTempBuffer đến bộ đệm chiều cao trong Lớp CTERRAIN, m\_heightData.
      * Kiểm tra hình 2.9 để xem một số bản đồ chiều cao hình thành sử dụng sự hình thành lỗi và thay đổi số dòng lỗi lặp đi lặp lại. Gần như chúng ta, chúng tôi chưa hoàn thành với thuật toán này chưa! Trong trường hợp bạn không để ý, cái bản đồ trong hình trước nhìn phi địa hình (mới từ). Chúng ta cần phải vượt qua bộ lọc xói mòn trên toàn bộ bản đồ



* + - * sau khi chúng tôi hình thành một fault mới để làm phẳng các giá trị mà chúng tôi có. Cái này quá trình rất nhiều, nếu không muốn nói là chính xác, giống như truyền một bộ lọc mờ qua một hình ảnh trong chương trình sơn yêu thích của bạn. Nếu nó giúp bạn hiểu giải thích sau đây, chỉ cần nghĩ về nó như thế.
      * Những gì chúng ta sẽ làm là áp dụng một bộ lọc FIR đơn giản, như đề xuất của Jason Shankel.1 Bộ lọc này nhằm mô phỏng sự xói mòn địa hình, điều thường xuyên xảy ra trong tự nhiên. (Bạn đã bao giờ xem một bộ của những ngọn núi trong tự nhiên trông giống như các bản đồ chiều cao trong Hình 2.9?) Chúng tôi sẽ lấy dữ liệu theo dải, thay vì lọc toàn bộ sơ đồ chiều cao cùng một lúc. Chức năng lọc trông như thế này:



* + - * Hàm này lấy một dải giá trị chiều cao và đi qua chúng đánh giá theo giá trị, với iStride chỉ định mức tăng các giá trị theo cho mỗi lần lặp trong vòng lặp. iStride cũng ra lệnh mà hướng đi vì chúng tôi sẽ lọc toàn bộ trường chiều cao từ trên xuống dưới, từ dưới lên trên, từ trái sang phải và phải sang trái. nhiều nhất dòng quan trọng của toàn bộ chức năng là dòng này:



* + - * Đây là dòng làm mờ/xói mòn(blurs/erodes). Các giá trị khác nhau cho fFilter ảnh hưởng đến làm mờ(blurring). 0.0f hoàn toàn không bị mờ và 1.0f là độ mờ thực sự mạnh. Thông thường, chúng tôi muốn các giá trị nằm trong khoảng 0.3f đến 0.6f, tùy thuộc vào bạn muốn địa hình bằng phẳng như thế nào. Bây giờ, ví dụ, hãy nói chúng tôi có giá trị bộ lọc là 0.25f và giá trị dải hiện tại(current band value) là 0.9f. Các phương trình trước đó sẽ trông như thế này:



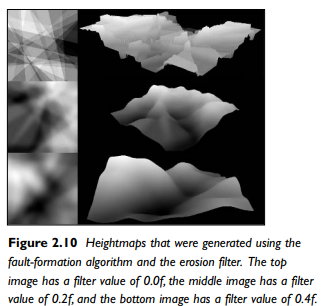
* + - * Sau khi chúng tôi thực hiện các tính toán ban đầu, phương trình trước đó sẽ đơn giản hóa xuống thế này:



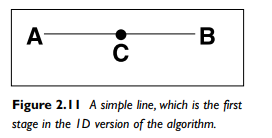
* + - * 0.675f là giá trị mới cho pixel heightmap mà chúng tôi đang làm mờ,nhưng bây giờ nó cần được nội suy với pixel trước nó. (Tốt cung cấp cho pixel đó giá trị là 0,87f.) Chúng tôi áp dụng bộ lọc làm mờ 0.25f giá trị cho pixel đó và thêm nó vào giá trị pixel không được nội suy cho pixel mà chúng tôi đang cố gắng tính toán:



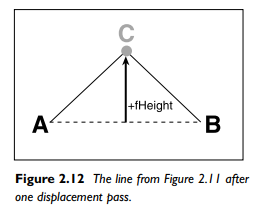
* + - * Thực hiện các phép tính cuối cùng, chúng ta có giá trị là 0,8925f. Vì thế, bạn thấy đấy, tất cả những gì chúng tôi đang thực sự làm ở đây là “pha trộn” một chút pixel với pixel hiện tại. Hãy xem Hình 2.10 để xem cách lọc trông như thế nào trên quy mô lớn hơn nhiều so với các hoạt động trên mỗi pixel mà chúng tôi trước đây đã nói về.
      * Chơi xung quanh với demo2\_2 một chút. Tôi đã tạo một khu vực mới của menu cho thao tác bản đồ chiều cao và bây giờ bạn có thể tạo bản đồ chiều cao mới một cách năng động. Nếu bạn tìm thấy một cái bạn thích, chỉ cần chọn Lưu hiện tại tùy chọn và sơ đồ chiều cao sẽ được lưu vào thư mục của chương trình. Khi bạn chọn tùy chọn Fault Formation, một hộp thoại sẽ mở ra và nhắc bạn nhập một giá trị để biết chi tiết. Giá trị này là một số nguyên giá trị, vì vậy hãy giữ các số trong phạm vi 0–100 cho việc này. Bây giơ là thời gian cho một số niềm vui dịch chuyển trung điểm



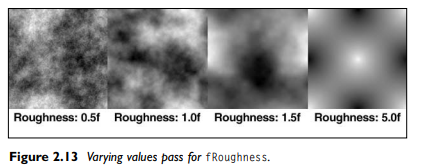
* + - Thuật toán midpoint displacement
      * Fault formation hoạt động tuyệt vời cho một cảnh nhỏ đẹp bao gồm một số những ngọn đồi nhỏ, nhưng nếu bạn muốn thứ gì đó hỗn loạn hơn thế thì sao, chẳng hạn như một dãy núi? Chà, đừng tìm đâu xa. Midpoint displacement là câu trả lời mà bạn đang tìm kiếm! Thuật toán này còn được gọi là plasma fractal và thuật toán hình vuông kim cương(The diamond-square algorithm). Tuy nhiên, midpoint displacement nghe hay hơn rất nhiều và nó mang lại cho người đọc (chính là bạn) một ý tưởng tốt hơn về những gì thực sự đang diễn ra trong toàn bộ quá trình này, vì vậy tôi sẽ dính vào thuật ngữ đó hầu hết thời gian.
      * Tất cả chúng tôi đang làm trong này về cơ bản, thuật toán đang lấy điểm giữa của một dòng và thay thế nó! Để tôi cung cấp cho bạn một chiều chạy xuyên qua. Nếu chúng ta có một đường thẳng đơn giản, chẳng hạn như AB trong Hình 2.11, chúng ta sẽ lấy trung điểm, biểu diễn là C trong hình, và di chuyển nó!



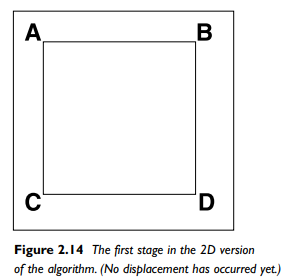
* + - * Bây giờ, chúng ta sẽ dịch chuyển điểm giữa của đường thẳng đó bằng một giá trị chiều cao, mà chúng ta sẽ gọi là fHeight (xem Hình 2.12). Chúng tôi sẽ làm cho nó tương đương với chiều dài của đoạn thẳng được đề cập và chúng tôi sẽ dịch chuyển điểm giữa bằng một phạm vi từ –fHeight/2 đến fHeight/2. (Chúng tôi muốn chia nhỏ dòng thành hai mỗi lần và chúng tôi muốn thay thế chiều cao của dòng ở đâu đó trong phạm vi đó.)



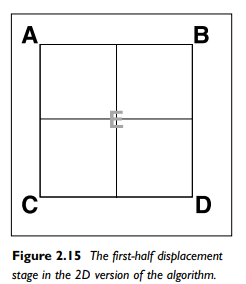
* + - * Sau lần vượt qua đầu tiên, chúng ta cần giảm giá trị của fHeight để đạt được độ nhám mà chúng ta mong muốn. Để làm điều này, chúng ta chỉ cần nhân fHeight với 2-fRoughness, trong đó fRoughness là một hằng số đại diện cho mong muốn độ gồ ghề của địa hình. Người dùng sẽ chỉ định giá trị cho fRoughness, vì vậy bạn cần biết một chút về các giá trị khác nhau mà bạn có thể đặt cho nó. Về mặt kỹ thuật, giá trị có thể là bất kỳ giá trị dấu phẩy động nào mà trái tim bạn mong muốn, nhưng kết quả tốt nhất là từ 0.25f đến 1.5f. Kiểm tra Hình 2.13 để có một chỉ báo trực quan về những mức độ nhám khác nhau có thể làm được.
      * Như bạn có thể thấy, giá trị bạn chuyển cho fRoughness ảnh hưởng lớn đến nhìn của bản đồ chiều cao. Các giá trị thấp hơn 1.0f tạo ra sự hỗn loạn địa hình, các giá trị 1.0f tạo ra giao diện khá “cân bằng” và các giá trị đều lớn hơn 1.0f tạo ra địa hình bằng phẳng. Bây giờ, hãy chuyển lời giải thích này sang chiều thứ hai.



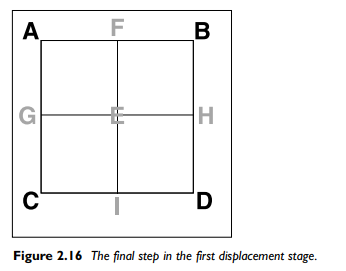
* + - * Giữ lời giải thích 1D trong đầu bạn liên tục khi chúng ta nói về cần thay đổi điều gì đối với phần giải thích 2D vì mọi khái niệm bạn chỉ cần đã học cho dòng đơn đó vẫn được áp dụng. Ngoại lệ là, thay vào đó tính toán điểm giữa cho một dòng, bây giờ chúng ta phải tính toán trung điểm của bốn dòng khác nhau, lấy trung bình cộng rồi cộng giá trị chiều cao ở giữa hình vuông. Hình 2.14 cho thấy chỗ trống hình vuông (ABCD) mà chúng ta bắt đầu với.



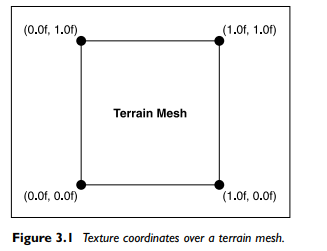
* + - * Như tôi đã nói một giây trước, chúng ta phải tính trung điểm cho cả bốn các đường thẳng (AB, BD, DC, CA). Điểm kết quả, E, phải trực tiếp trong chính giữa hình vuông. Sau đó, chúng tôi thay thế E bằng cách lấy trung bình của Các giá trị chiều cao của A, B, C và D, sau đó chúng tôi thêm một giá trị ngẫu nhiên vào phạm vi từ –fHeight/2 đến fHeight/2. Điều này dẫn đến hình ảnh hiển thị trong Hình 2.15.



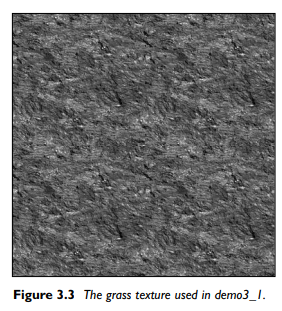
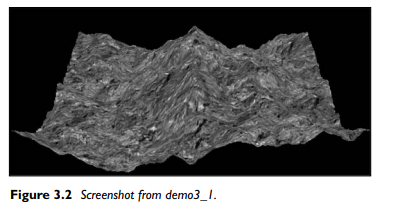
* + - * Đó chỉ là nửa đầu của giai đoạn dịch chuyển đầu tiên. Bây giờ chúng ta phải tính các giá trị chiều cao cho từng trung điểm mà chúng ta được tìm thấy trước đó. Tuy nhiên, điều này tương tự như những gì chúng tôi đã làm trước đây; chúng tôi chỉ lấy trung bình các giá trị độ cao của các đỉnh xung quanh và thêm một giá trị độ cao ngẫu nhiên trong khoảng từ –fHeight/2 đến fHeight/2. bạn kết thúc với một hình vuông như trong Hình 2.16.
      * Sau đó, bạn quay lại tập hợp các hình chữ nhật tiếp theo và thực hiện cùng một quy trình. Tuy nhiên, nếu bạn hiểu lời giải thích 1D, thì bạn nhất định phải hiểu giải thích 2D và mã đi kèm, demo2\_2, được tìm thấy trên CD trong Code\Chương 2\demo2\_2. Biên dịch thông tin, như thường lệ, được cung cấp dưới dạng tệp văn bản trong bản trình diễn danh mục. Đi kiểm tra bản demo. Điều khiển giống như lần trước thời gian (xem Bảng 2.1 để biết lời nhắc), nhưng lần này, khi bạn bấm vào Độ dịch chuyển điểm giữa cho trường Chi tiết, bạn muốn các giá trị trong trường phạm vi từ 0 (địa hình thực sự hỗn loạn) đến 150 (địa hình đơn giản). Chúc vui vẻ!



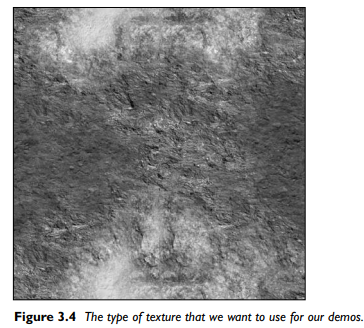
1. Chương 3: Texturing Terrain
   * + bây giờ bạn đã có phần giới thiệu về cách tạo một địa hình đơn giản lưới, bạn cần biết cách thêm chi tiết vào cái thứ nhàm chán đó bằng cách sử dụng bản đồ kết cấu(texture map). Tôi sẽ tiếp tục cuộc thảo luận này về kết cấu đơn giản và đi thẳng vào vấn đề để chúng ta có thể bắt đầu với những thứ thực sự thú vị (thuật toán địa hình-the terrain algorithms). tôi sẽ nghỉ việc lãng phí không gian bây giờ và chỉ cho bạn biết những gì bạn sẽ được họctrong chương này:
       - Cách áp dụng bản đồ kết cấu đơn mẫu lớn(large single-pattern texture map) cho địa hình lưới.
       - Cách tạo bản đồ kết cấu phức tạp theo thủ tục (complex texture map) bằng cách sử dụng "tiles" địa hình khác nhau.
       - Cách thêm kết cấu chi tiết vào địa hình để thêm nhiều chi tiết hơn vào kết cấu đã tạo trước đó.
   1. Simple Texture Mapping
      * Chúng ta sẽ bắt đầu với một số ánh xạ kết cấu đơn giản. Bạn sẽ tìm hiểu cách “kéo dài” một kết cấu trên toàn bộ lưới địa hình. Hầu hết vào thời điểm đó, kỹ thuật này có vẻ rất tệ, tất nhiên, trừ khi bạn có một bản đồ kết cấu thực sự tốt, đó là những gì chúng ta sẽ làm việc trên trong phần tiếp theo. Điều quan trọng bây giờ là bạn học cách kéo giãn kết cấu mà không cần quan tâm đến kết quả cuối cùng sẽ như thế nào.
      * Để kéo dài một kết cấu duy nhất trên toàn cảnh, chúng ta sẽ tạo mọi đỉnh trong cảnh nằm trong phạm vi 0,0f–1,0f (các phạm vi tiêu chuẩn cho tọa độ kết cấu). Làm điều này thậm chí còn dễ dàng hơn nó có vẻ. Để bắt đầu, hãy nhìn vào Hình 3.1.
      * Như Hình 3.1 cho thấy, góc dưới bên trái của lưới địa hình (đối với mục đích ví dụ, chúng tôi sẽ chọn độ phân giải sơ đồ chiều cao là 256 × 256), (0,0) sẽ có tọa độ kết cấu là (0,0f, 0,0f) và phía trên bên trái góc của địa hình (255, 255), sẽ có tọa độ kết cấu là (1.0f, 1.0f). Về cơ bản, tất cả những gì chúng ta cần làm là tìm xem chúng ta đang ở đỉnh nào hiện đang hiển thị và chia nó cho độ phân giải của sơ đồ chiều cao. (Làm như vậy sẽ tạo ra các giá trị trong phạm vi mà chúng ta muốn, 0,0f–1,0f, mà không cần chúng tôi bước qua ranh giới của chúng tôi. Điều này rất quan trọng cần lưu ý bởi vì chúng tôi làm muốn bước ra khỏi phạm vi đã đề cập trước đó trong phần sau.) Trước khi kết xuất từng đỉnh, chúng ta cần tính toán ba điều: giá trị kết cấu cho x, z và z+1, mà tôi sẽ gọi là fTexLeft, fTexBottom và fTexTop, tương ứng. Đây là cách chúng tôi tính toán các giá trị:



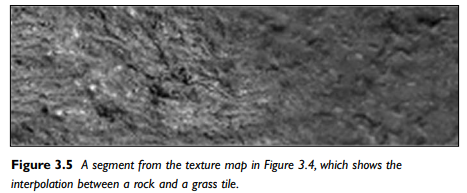
* + - Và để nghĩ rằng bạn nghĩ rằng điều này sẽ được khó khăn! Dù sao, chúng ta cần để thực hiện các phép tính trước đó cho từng đỉnh mà chúng tôi hiển thị và sau đó gửi tọa độ kết cấu tới API kết xuất của chúng tôi. Khi nào chúng ta hiển thị đỉnh (x, z), chúng tôi gửi (fTexLeft, fTexBottom) làm kết cấu của chúng tôi tọa độ và khi chúng tôi kết xuất (x, z+1), chúng tôi gửi (fTexLeft, fTexTop) như tọa độ kết cấu của chúng tôi. Kiểm tra Hình 3.2 và demo3\_1 trên CD trong Code\Chapter 3\demo3\_1 để xem thành quả lao động của bạn.
    - Ảnh chụp màn hình có nhiều chi tiết hơn phong cảnh của chúng ta trong Chương 2, “Terrain 101” (tuy nhiên, lưu ý rằng tôi đã loại bỏ bóng đổ), nhưng nó khó để phân biệt hình thức thực tế của cảnh quan. Kéo giãn một kết cấu đơn giản (xem Hình 3.3), ngay cả khi kết cấu được sử dụng trong demo3\_1 ở mức độ phân giải khá cao, không nắm bắt được lượng chi tiết mà chúng tôi muốn có trong bản đồ kết cấu của chúng tôi.



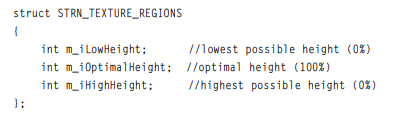
* + - Chúng tôi cần thêm chi tiết. Chúng tôi muốn có một bản đồ kết cấu tương tự như trong Hình 3.4, được tạo thủ tục bằng cách sử dụng một loạt kết cấu "gạch" (trong trường hợp này là đất, cỏ, đá và tuyết).
    - Xem bao nhiêu chi tiết được hiển thị trong kết cấu của Hình 3.4? Kết cấu của hình này giúp phân biệt các vùng núi cao với vùng thấp khu vực đồng bằng tốt hơn nhiều so với kết cấu cỏ đơn lẻ được hiển thị trong Hình 3.3. Bạn cần biết cách tạo ra một kết cấu thực sự thú vị như một hiển thị ở đây. Đọc tiếp!



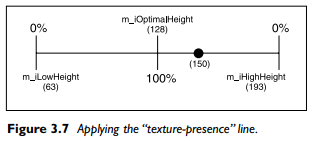
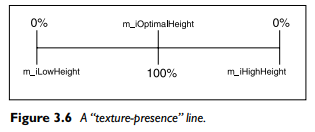
* 1. Procedural Texture Generation
     + Tạo kết cấu theo thủ tục là một kỹ thuật thú vị và hữu ích. bổ sung cho bất kỳ động cơ địa hình. Sau khi chúng tôi hoàn thành kết cấu thủ tục của chúng tôi trình tạo, chúng tôi sẽ yêu cầu người dùng tải một loạt từ hai đến bốn ô của sự lựa chọn của mình. Sau đó, chúng ta sẽ gọi hàm tạo kết cấu của mình. (Tất cả những gì người dùng phải biết là kích thước của kết cấu mà anh ta muốn được tạo ra.) Thế là xong! Làm thế nào để chúng ta bắt đầu tạo thế hệ kết cấu của mình chức năng? Đầu tiên, bạn cần biết mục tiêu thực sự của chúng ta ở đây là gì. Chúng tôi là sẽ sử dụng bản đồ chiều cao của địa hình để tạo kết cấu sẽ trùng với nó. Chúng tôi sẽ đi qua từng pixel của bản đồ kết cấu của chúng tôi, tìm chiều cao tương ứng với pixel đó và tìm ra từng sự hiện diện của ô kết cấu tại pixel đó. (Mỗi ô có cấu trúc “vùng” xác định các khu vực ảnh hưởng của nó.) Rất hiếm khi một ô hiển thị 100% tại một pixel, vì vậy chúng ta cần “kết hợp” ô đó với phần còn lại của ô khác gạch (nội suy các giá trị màu RGB). Kết quả sẽ trông giống như một cái gì đó giống như trong Hình 3.5, nơi bạn có thể thấy phép nội suy sẽ như thế nào trông giống như giữa cỏ và gạch đá.



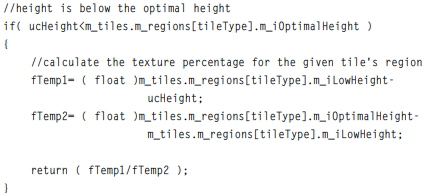
* 1. The Region System
     + Để bắt đầu mã hóa quy trình đã đề cập trước đó, chúng ta cần bắt đầu bằng cách tạo một cấu trúc để giữ thông tin vùng cho mỗi ô. Một vùng, như được áp dụng ở đây, là một chuỗi gồm ba giá trị xác định kích thước của một ô sự hiện diện trên phạm vi giá trị chiều cao của chúng tôi (0–255). Đây là cấu trúc mà tôi đã tạo:



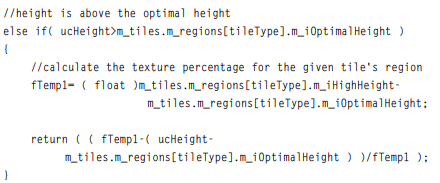
* + - Một lời giải thích về những gì mỗi giá trị làm được thực hiện tốt nhất bởi kiểm tra Hình 3.6.
    - Để giải thích, chúng tôi sẽ tạo m\_iLowHeight tương đương với 63 và m\_iOptimalHeight tương đương với 128. Tính giá trị cho m\_iHighHeight yêu cầu một số phép toán đơn giản. Chúng tôi muốn trừ m\_iLowHeight từ m\_iOptimalHeight. Sau đó, chúng tôi muốn thêm m\_iOptimalHeight thành kết quả của thao tác trước đó. Chúng tôi có giới hạn được đặt (thấp: 63, tối ưu: 128, cao: 193), vì vậy hãy thay thế các giới hạn đó các giá trị biên trong Hình 3.6. Bây giờ hãy tưởng tượng rằng chúng ta đang cố gắng tìm ra mức độ hiện diện của ô hiện tại ở độ cao, giả sử, 150. Hãy tưởng tượng giá trị đó sẽ ở đâu trên dòng trong Hình 3.6, có tính đến các ranh giới mà chúng tôi đã tạo ra. Để tiết kiệm cho bạn rắc rối của việc cố gắng tìm ra nó, hãy xem Hình 3.7.



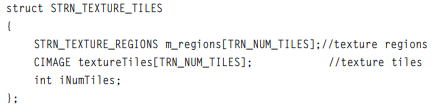
* + - Như bạn có thể thấy qua hình ảnh, sự hiện diện của kết cấu ở độ cao đó (150) là khoảng 70 phần trăm. Bây giờ chúng ta đã biết nhiều thông tin đó, Chúng ta làm gì với nó? Chà, chúng tôi trích xuất bộ ba RGB từ hình ảnh của kết cấu và nhân nó với 0,7f. Kết quả là chúng ta muốn bao nhiêu kết cấu ở pixel hiện tại.
    - Chúng ta cần tạo một hàm sẽ tính phần trăm khu vực cho chúng tôi. Chức năng này khá đơn giản. Nó đòi hỏi hai bài kiểm tra tầm thường để xem liệu độ cao nhất định có thực sự nằm trong ranh giới của khu vực hay không; nếu không, thoát khỏi chức năng. Tiếp theo, chúng ta cần tìm ra nơi chiều cao nằm trong vùng. Nó ở dưới giá trị tối ưu, trên nó, hoặc tương đương với giá trị tối ưu? Trường hợp tầm thường là nếu chiều cao là tương đương với giá trị tối ưu; nếu đúng như vậy, thì ô hiện tại có sự hiện diện của kết cấu là 100 phần trăm ở pixel hiện tại và chúng tôi không cần lo lắng về nội suy ở tất cả.
    - Nếu chiều cao dưới giá trị tối ưu, chúng ta cần giảm giá trị đã cho các giá trị thành một phân số đơn giản. Để làm điều này, chúng tôi lấy chiều cao nhất định và trừ nó bằng ranh giới thấp cho khu vực. Sau đó, chúng tôi lấy giá trị ranh giới tối ưu và trừ nó bằng ranh giới thấp. Sau đó chúng tôi chia kết quả từ phép tính đầu tiên cho kết quả từ phép tính thứ hai và BAM! Chúng tôi có tỷ lệ hiện diện của chúng tôi!
    - Đây là mã cho những gì vừa được thảo luận:



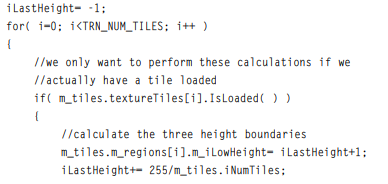
* + - Trường hợp cuối cùng là nếu chiều cao đã cho nằm trên ranh giới tối ưu. Các tính toán cho trường hợp này phức tạp hơn một chút so với khi chiều cao thấp hơn ranh giới, nhưng chúng vẫn không cứng lắm. Cái này giải thích ở dạng mã dễ nhìn hơn nhiều so với ở dạng văn bản, vì vậy đây là mã:

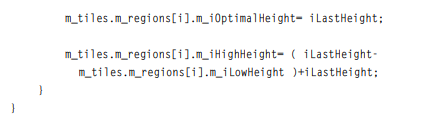


* + - Về lý thuyết, các tính toán về cơ bản giống như chúng đã tính toán trường hợp thấp hơn chiều cao tối ưu, ngoại trừ việc chúng tôi phải có khả năng lấy các giá trị xuống một phân số có ý nghĩa vì 100 phần trăm thấp hơn chiều cao, thay vì cao hơn chiều cao. Đó là nó
  1. The Tile System
     + Được rồi, bây giờ bạn đã biết cách lấy sự hiện diện của họa tiết cho một họa tiết gạch và một pixel kết cấu. Bây giờ bạn cần áp dụng mọi thứ bạn vừa đã học cách tính đến cả bốn ô kết cấu và tạo toàn bộ bản đồ kết cấu. Tuy nhiên, điều này dễ hơn nhiều so với âm thanh, vì vậy đừng hiểu choáng ngợp!
     + Trước tiên, chúng ta cần tạo cấu trúc ô kết cấu có thể quản lý tất cả gạch kết cấu của chúng tôi. Chúng tôi không cần nhiều thông tin cho mỗi ô; tất cả chúng tôi cần là một nơi để tải kết cấu vào và một cấu trúc vùng cho từng viên gạch. Chúng tôi cũng sẽ muốn theo dõi tổng số ô được tải. Với tất cả những yêu cầu đó trong tâm trí, tôi đã tạo raCấu trúc STRN\_TEXTURE\_TILES, giống như sau:

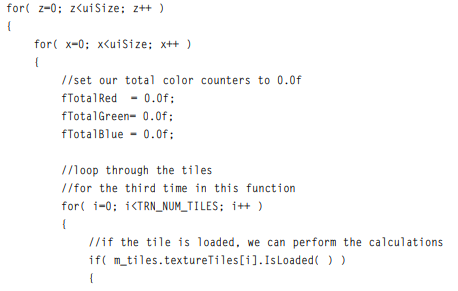


* + - Tiếp theo, bạn cần một số chức năng quản lý gạch kết cấu. Tôi có chức năng tải và dỡ một ô, cùng với chức năng dỡ bỏ tất cả các ô cùng một lúc. Các chức năng này là tầm thường để thực hiện, vì vậy tôi sẽ không hiển thị một đoạn trích nào ở đây. Chỉ cần nhìn vào mã nếu bạnthú vị. Ngoài ra, bạn đã sẵn sàng viết mã chức năng tạo kết cấu!
    - Để bắt đầu chức năng tạo, chúng ta cần tính xem có bao nhiêu ô thực sự được tải. (Chúng tôi muốn người dùng có thể tạo kết cấu mà không tải cả bốn ô.) Sau khi thực hiện xong, chúng ta cần lặp lại qua các ô để tìm ra ranh giới khu vực cho từng ô. (Chúng tôi muốn các vùng ô vuông được đặt cách đều nhau trên 0–255 phạm vi). Đây là cách tôi đã làm điều này:

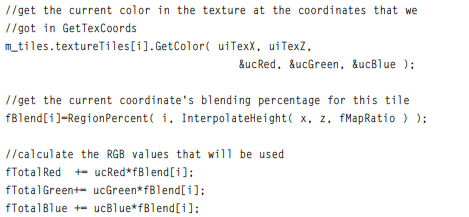




* + - Điều duy nhất có vẻ hơi kỳ lạ ở đây là phân đoạn cuối cùng nơi chúng tôi tính toán m\_iHighHeight và thậm chí điều đó sẽ không giống như vậy lẻ bởi vì chúng tôi đã giải thích nó trước đó. (Nếu nó trông kỳ lạ, hãy tham khảo lạiphần đầu của phần này, nơi tôi giải thích các ranh giới khu vực.)
  1. Creating the Texture Data
     + Bây giờ là lúc để tạo dữ liệu kết cấu thực tế. Để làm được điều này, chúng ta cần phải tạo ba vòng lặp for khác nhau: một cho trục Z của bản đồ kết cấu, một cho trục X và một đi qua mỗi ô. (Cái này sẽ vòng lặp gạch thứ ba trong chức năng này.) Chúng ta cũng cần tạo ba biến sẽ duy trì tổng số màu đỏ, xanh lá cây và xanh dương hiện tại đang chạy các thành phần cho mỗi pixel mà chúng tôi đang tính toán. Đây là những gì bắt đầu tạo kết cấu thực tế sẽ giống như:



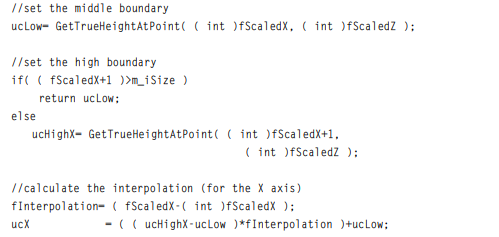
* + - Tiếp theo, chúng ta cần trích xuất các giá trị RGB từ kết cấu (tại pixel hiện tại) vào các biến char không dấu RGB tạm thời của chúng tôi. Một lần thế là xong, chúng ta cần tìm ra sự hiện diện của ô hiện tại ở pixel hiện tại (sử dụng hàm mà chúng ta đã tạo trước đó), nhân biến RGB tạm thời theo kết quả và thêm nó vào tổng số RGB của chúng tôi quầy. Bây giờ chúng ta cần đưa lời giải thích trước đó vào mã:



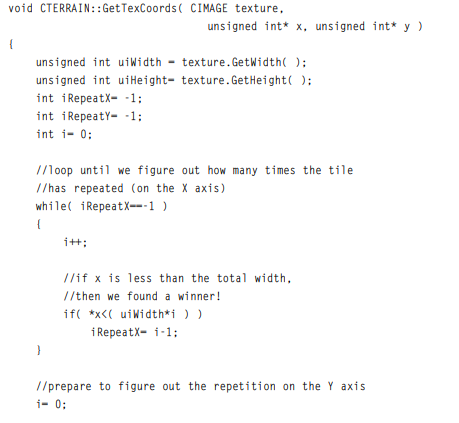
* + - Sau khi chúng tôi đã lặp qua tất cả bốn ô, sau đó chúng tôi đặt màu cho pixel trong kết cấu mà chúng tôi đang tạo, sau đó làm lại toàn bộ điều cho pixel tiếp theo. Khi chúng tôi hoàn thành việc tạo các giá trị màu cho kết cấu, chúng tôi tạo kết cấu để sử dụng với API đồ họa và chúng ta đã sẵn sàng!
  1. Improving the Texture Generator
     + Được rồi, tôi đã nói dối. Chúng tôi chưa sẵn sàng. Chức năng tạo kết cấu của chúng tôi có một một vài vấn đề ở dạng hiện tại của nó. Những vấn đề này như sau:
       - Chúng tôi chỉ có thể tạo một kết cấu có độ phân giải bằng hoặc thấp hơn, sơ đồ chiều cao.
       - Nếu chúng tôi khắc phục vấn đề đó, thì chúng tôi chỉ có thể tạo kết cấu với một độ phân giải bằng hoặc thấp hơn độ phân giải của ô kết cấu nhỏ nhất của chúng tôi.
     + Tuy nhiên, cả hai vấn đề này đều tương đối dễ khắc phục. Hãy bắt đầu với vấn đề độ phân giải bản đồ chiều cao.
     + Getting Rid of the Heightmap Resolution Dependency
       - Chúng ta cần cho phép người dùng chọn bất kỳ kích thước kết cấu nào mà họ muốn. (Tốt, hầu như bất kỳ kích thước. Chúng tôi muốn các thứ nguyên là lũy thừa của 2.) Ngay từ đầu chức năng tạo kết cấu của chúng tôi, trước khi chúng tôi nhập một loạt các vòng lặp for, chúng tôi cần tìm ra sơ đồ chiều cao cho pixel bản đồ kết cấu tỷ lệ, có thể được thực hiện như thế này:

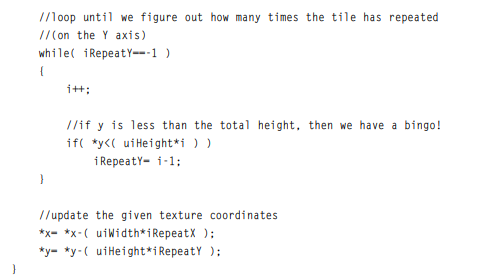


* + - * Sau đó, chúng ta cần tạo một hàm sẽ nội suy các giá trị mà chúng tôi trích xuất từ sơ đồ chiều cao. Chúng ta sẽ chia chức năng nội suy này thành hai phần: một phần cho trục X và một phần cho trục Z. Chúng tôi sẽ sau đó lấy giá trị trung bình của kết quả cho cả hai phần, đó là giá trị cho giá trị chiều cao nội suy của chúng tôi. Đây có thể không phải là cách tốt nhất để đi về mọi thứ, nhưng nó hoạt động, và nó hoạt động nhanh chóng!
      * Đối với chức năng này, chúng ta cần ba đối số. Hai đối số đầu tiên là tọa độ (x, z) chưa chia tỷ lệ mà chúng tôi đang lấy thông tin. Điều này sẽ khá cao, và rất có thể, vượt ra ngoài phạm vi của sơ đồ chiều cao. Đối số thứ ba là biến có tính toán của chúng tôi chiều cao so với tỷ lệ pixel bản đồ kết cấu (fMapRatio). Bên trong chức năng, chúng tôi sẽ chia tỷ lệ tọa độ (x, z) đã cho theo biến tỷ lệ và sẽ sử dụng những người cho hầu hết các chức năng. Để tính toán phép nội suy dọc theo trục X, chúng tôi sẽ làm điều này:

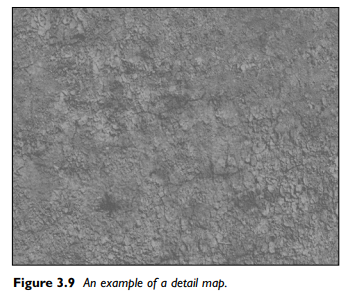
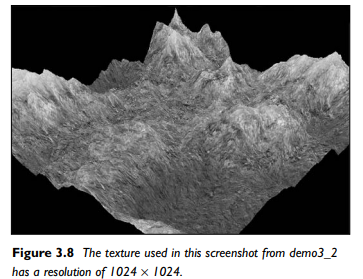


* + - * Như bạn có thể thấy, điều đầu tiên mà chúng tôi làm là lấy chiều cao thấp. Sau đó chúng tôi kiểm tra xem chiều cao tiếp theo trên sơ đồ chiều cao có bằng nhau không sơ đồ chiều cao. Nếu không, thì chúng ta phải hài lòng với giá trị thấp. Nếu chiều cao tiếp theo nằm trên sơ đồ chiều cao, thì chúng ta có thể lấy nó và lấy sẵn sàng để nội suy hai giá trị. Chúng tôi nhận được sự khác biệt giữa giá trị x được chia tỷ lệ dấu phẩy động và giá trị x được chia tỷ lệ char không dấu.
      * (Nó sẽ có mức độ chính xác thấp hơn, điều này sẽ xác định số lượng của phép nội suy mà chúng ta sẽ sử dụng.) Trong phép tính tiếp theo, chúng ta tính nội suy dọc theo trục X. Sau đó chúng tôi làm điều tương tự cho trục Z, cộng kết quả từ cả hai phép tính và chia cho 2. Thats tất cả để có nó!
    - Getting Rid of the Tile Resolution Dependency
      * Được rồi, chúng tôi gần như ở đó. Chúng ta chỉ còn một điều nữa để tìm ra, và đó là cách chúng ta có thể loại bỏ ranh giới kích thước ô được đặt khi người dùng. Giải pháp rõ ràng đến mức bạn có thể thắc mắc, "Tại sao tôi không nghĩ về điều đó?" Chà, tin tôi đi, tôi đã mất một thời gian dài để tìm ra giải pháp, vì vậy đừng cảm thấy tồi tệ. Tất cả những gì chúng ta cần làm là lặp lại ngói! Tôi đã tạo một chức năng đơn giản sẽ cung cấp cho chúng ta kết cấu “mới” tọa độ sẽ lặp lại kết cấu của chúng tôi cho chúng tôi. Đây là chức năng:





* + - * Hầu hết chức năng này bao gồm hai vòng lặp while với mục tiêu chính là chỉ cố gắng tìm ra kết cấu đã lặp lại bao nhiêu lần trước khi nó đạt đến tọa độ đã được đưa ra làm đối số. Sau đó điều đó đã được tìm ra, chúng tôi chỉ thu nhỏ các tọa độ đã cho để rằng chúng trở lại trong phạm vi giá trị của kết cấu. (Chúng tôi không muốn cố gắng trích xuất thông tin hoàn toàn nằm ngoài phạm vi của kết cấu. Điều đó sẽ gây ra lỗi và lỗi rất tệ.)
      * Đó là nó! Chức năng tạo kết cấu của chúng tôi hiện đã hoàn tất! Kiểm tra ra Hình 3.8. Trong bản trình diễn, bạn sẽ nhận thấy một trường mới trong menu được gọi là Bản đồ kết cấu. Trong trường này, bạn có thể tạo một kết cấu mới của một độ phân giải cao hơn hoặc lưu kết cấu hiện tại vào thư mục của bản trình diễn. Nói về bản demo, bạn có thể thấy tất cả công việc khó khăn của mình trong bản demo3\_2 trên đĩa CD trong Code\Chapter 3\demo3\_2. Chỉ cần mở không gian làm việc cho bản trình diễn trong Microsoft Visual C++ và bắt đầu vui vẻ!
  1. Using Detail Maps
     + 1024 × 1024 là một lượng dữ liệu khá lớn để đạt được số lượng chi tiết mà chúng tôi muốn có trong kết cấu của mình. phải có cái khác cách để đạt được chi tiết mong muốn của chúng tôi mà không lãng phí tài nguyên. Tốt, không nghĩ nữa! Một cách hay để thêm nhiều chi tiết hơn vào phong cảnh của bạn là bằng cách sử dụng một bản đồ chi tiết. Bản đồ chi tiết là một kết cấu thang độ xám giống như bản đồ trong Hình 3.9 được lặp lại nhiều lần trên một cảnh quan và thêm các sắc thái mát mẻ, chẳng hạn như vết nứt, va chạm, đá và những thứ thú vị khác.



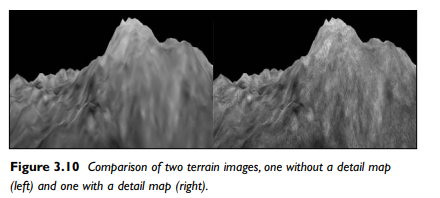
* + - Thêm hỗ trợ bản đồ chi tiết vào công cụ địa hình của bạn là một quá trình đơn giản. Thêm chức năng quản lý bốc/ dỡ chi tiết bản đồ, một chức năng cho phép người dùng quyết định số lần anh ta muốn bản đồ lặp lại trên toàn cảnh, và sau đó tất cả những gì bạn phải làm làm là chỉnh sửa mã hiển thị của bạn một chút. Quyết định khó khăn nhất là quyết định xem có sử dụng đa kết cấu phần cứng hay chỉ thực hiện hai lần kết xuất riêng biệt. Bởi vì lưới địa hình có thể trở nên khá lớn, tốt nhất bạn đặt cược là dính vào đa kết cấu phần cứng. Triển khai phần cứng đa kết cấu nằm ngoài phạm vi của cuốn sách này, nhưng nếu bạn không biết làm thế nào để làm điều đó với API đồ họa của bạn, đó là một điều đơn giản để thực hiện, và bạn nên học nó. Trong trường hợp bạn không biết một nơi tốt để học API, hãy xem Lập trình trò chơi OpenGL (Astle/Hawkins) hoặc Lập trình trò chơi hiệu ứng đặc biệt với DirectX 8.0 (McCuskey), cả hai được xuất bản bởi Premier Press—một nhà xuất bản tuyệt vời nếu tôi tự nói như vậy!
    - Để chỉnh sửa mã hiển thị của bạn, chỉ cần đặt kết cấu màu cơ bản (màu chẳng hạn như chúng tôi đã tạo trước đó) cho đơn vị kết cấu đầu tiên, rồi sau đó đặt kết cấu chi tiết của bạn thành đơn vị kết cấu thứ hai. nhớ làm thế nào các tọa độ kết cấu cho kết cấu màu được tính toán như thế này?



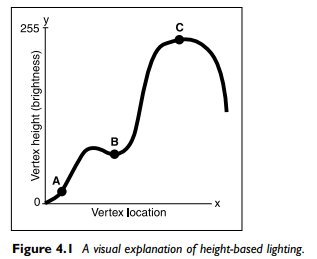
* + - Chà, chúng ta chỉ cần thực hiện một sửa đổi nhỏ đối với những tính toán đó để có được tọa độ kết cấu cho kết cấu chi tiết của chúng tôi:



* + - m\_iRepeatDetailMap là số lần người dùng muốn bản đồ chi tiết lặp lại trên toàn cảnh. Phần khó nhất của ánh xạ chi tiết là thiết lập đa kết cấu với API đồ họa của bạn, nhưng nếu bạn đang sử dụng OpenGL, tôi đã thiết lập mọi thứ cho bạn. (Vâng, tôi biết tôi là một chàng trai tốt, và sinh nhật của tôi là vào ngày 11 tháng 3 nếu bạn cảm thấy cần phải trả ơn tôi!) xem ảnh hưởng mà bản đồ chi tiết có thể có trên địa hình của bạn, nhìn vào sự khác biệt giữa một địa hình sử dụng kết cấu thủ tục 256 × 256 không có bản đồ chi tiết (bên trái Hình 3.10) và địa hình sử dụng cùng một kết cấu 256 × 256, ngoại trừ bản đồ chi tiết (phía bên phải của Hình 3.10).
    - Xem hình ảnh bên phải có bao nhiêu chi tiết? Phần tốt nhất là việc thêm số lượng chi tiết phong phú đó thật đơn giản. Thủ tục thanh toán demo3\_3 trên CD trong Code\Chapter 3\demo3\_3, hiển thị mã bản đồ chi tiết mới đang hoạt động. Các điều khiển thay đổi duy nhất từ phần còn lại của các bản demo là T tắt ánh xạ chi tiết và D tắt lập bản đồ chi tiết trở lại. (Ánh xạ chi tiết được bật theo mặc định.)



1. Chương 4: Lighting Terrain
   * + kết cấu địa hình mang lại một mức độ chi tiết mới vào của chúng tôi địa hình và chiếu sáng địa hình sẽ mang đến một mức độ chân thực hoàn toàn mới. Câu hỏi đặt ra là: Làm cách nào để chiếu sáng địa hình nhanh nhất có thể trong khi vẫn giữ được mức độ chân thực cao? Chà, tất cả các kỹ thuật mà tôi sẽ dạy cho bạn đều là những cách chiếu sáng địa hình nhanh (nếu khó). Tôi sẽ không đi sâu vào các thuật toán chiếu sáng toàn cầu phức tạp (mặc dù tôi sẽ chỉ cho bạn một số nơi mà bạn có thể lấy thông tin về một số thuật toán đó) bởi vì ánh sáng địa hình thực tế có thể tự nó bao phủ toàn bộ một cuốn sách. Như đã nói, đây là chương trình nghị sự cho chương này:
       - Chiếu sáng theo chiều cao
       - ánh sáng phần cứng
       - Áp dụng bản đồ ánh sáng cho địa hình
       - Thuật toán chiếu sáng dốc cực hay
     + Tôi sẽ giữ cuộc thảo luận về ánh sáng này thật ngắn gọn vì tôi biết rằng bạn (hoặc nếu bạn không, thì tôi biết là tôi có) muốn bắt đầu với các thuật toán địa hình thú vị mà tôi sẽ trình bày trong ba chương tiếp theo. Hãy bắt đầu đi!
   1. Height-Based Lighting
      * Chiếu sáng theo chiều cao rất đơn giản và không thực tế, nhưng nó là một kiểu chiếu sáng, vì vậy tôi nghĩ rằng ít nhất tôi nên trình bày ngắn gọn về nó. Chúng tôi đã sử dụng ánh sáng dựa trên chiều cao trong tất cả các bản trình diễn trong Chương 2, “Địa hình 101”, vì vậy bạn đã sử dụng nó trước đây ngay cả khi bạn không biết nó
      * Chiếu sáng dựa trên chiều cao chính là như vậy—chiếu sáng dựa trên chiều cao của một đỉnh. Đỉnh cao (dựa trên dữ liệu độ cao từ bản vá địa hình dữ liệu chiều cao) sáng hơn các đỉnh thấp và đó là tất cả những gì cần làm. Tất cả những gì chúng ta cần làm là sử dụng hàm GetTrueHeightAtPoint (thành viên của lớp CTERRAIN) để trích xuất độ sáng của pixel tại vị trí (x, z) hiện tại (giá trị sẽ nằm trong khoảng 0–255) từ bản đồ chiều cao và đó là giá trị độ sáng của chúng tôi. Nó đơn giản mà! Hình 4.1 củng cố khái niệm này.



* + - Trong hình, đỉnh A sẽ gần như đen, đỉnh B sẽ hơi sáng hơn và đỉnh C sẽ được chiếu sáng hoàn toàn (màu trắng). Đến đây rồi—một lời giải thích toàn bộ về ánh sáng dựa trên chiều cao trong ba đoạn văn và một con số!
    - Bây giờ bạn đã biết ánh sáng dựa trên chiều cao là gì, nhưng làm thế nào để bạn tính toán các giá trị ánh sáng bên trong mã của bạn? Chà, nó khá đơn giản, vì bạn đã tải một bản đồ chiều cao. Ví dụ: giả sử bạn đang cố gắng để tính toán độ sáng cho đỉnh (157, 227) trong địa hình của bạn. Tốt, độ sáng của đỉnh sẽ chỉ đơn giản là giá trị chiều cao mà bạn trích xuất từ sơ đồ chiều cao.



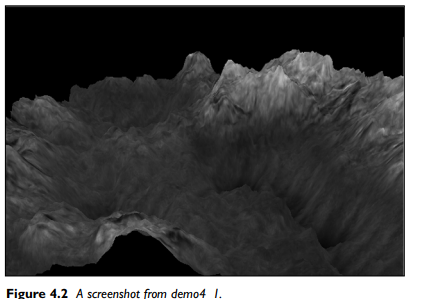
* + - ucShade là biến mà chúng tôi lưu trữ giá trị chiếu sáng của mình và GetTrueHeightAtPoint trích xuất thông tin từ sơ đồ chiều cao của chúng tôi, tại đỉnh (157, 227) trong trường hợp này, nằm trong khoảng từ 0 (tối) đến 255 (sáng). Bây giờ, hãy thêm một số màu sắc cho ánh sáng của chúng ta!
  1. Coloring the Light Source
     + Không phải lúc nào chúng ta cũng muốn màu chiếu sáng của mình là thang độ xám (đen đến trắng). Hầu hết thời gian, chúng tôi muốn ánh sáng của chúng tôi được tô màu cho các tình huống khác nhau. Chẳng hạn, nếu đó là một buổi tối không mây, người dùng sẽ trải qua một hoàng hôn đẹp, vì vậy chúng tôi muốn màu sắc ánh sáng của chúng tôi có màu cam, hồng hoặc tím. Chúng ta cần tạo một vectơ cho thông tin màu sắc ánh sáng và một chức năng đơn giản sẽ thiết lập màu sắc của ánh sáng. (Chúng tôi muốn các giá trị chiếu sáng nằm trong phạm vi của 0,0f–1,0f. Bạn sẽ tìm ra lý do tại sao trong giây lát.) Sau khi chúng tôi hoàn thành việc đó, chúng ta có thể lấy giá trị ánh sáng mà chúng ta đã truy xuất trước đó và nhân lên theo từng giá trị trong vectơ màu ánh sáng RGB bằng phương trình này:



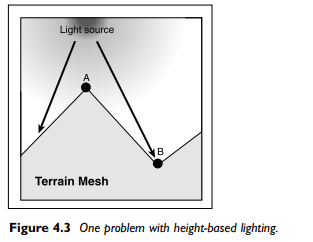
* + - Bây giờ, sử dụng phương trình đó, chúng ta có thể áp dụng nó để tìm ra màu RGB các thành phần, sau đó gửi chúng đến API hiển thị:



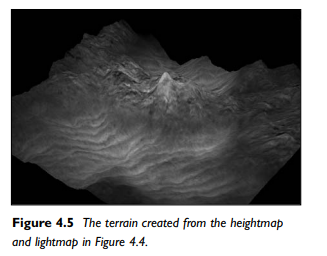
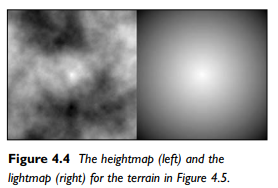
* + - ucShade là giá trị độ sáng mà chúng tôi đã tính toán trước đó và vecLightColor là màu ánh sáng của chúng ta. Bây giờ hãy xem demo4\_1 (trên đĩa CD dưới Code\Chapter 4\demo4\_1) và Hình 4.2. nếu bạn cần xem lại các điều khiển của bản trình diễn, hãy xem Bảng 4.1.
    - Khi bạn nhìn vào hình, bạn nhận thấy rằng các khu vực bên dưới của địa hình khá tối và các khu vực cao của địa hình sáng sủa. Đây chính xác là những gì hệ thống chiếu sáng dựa trên chiều cao thực hiện: Các khu vực cao sáng sủa, và các khu vực thấp là bóng tối. Vấn đề với thuật toán này là gì? Đầu tiên, nó là vô cùng phi thực tế. Thuật toán này không tính đến



* + - rằng mặt trời có thể chiếu trực tiếp vào một điểm “ngâm” trong địa hình (một khu vực có chiều cao thấp), sẽ làm cho khu vực đó rất sáng sủa. Vấn đề này được thể hiện trong Hình 4.3.



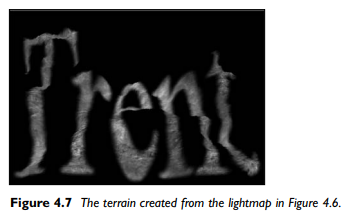
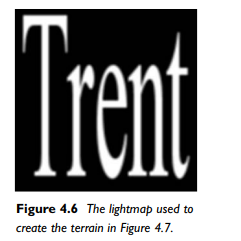
* + - Bạn thấy đấy, trong Hình 4.3, mặt trời đi tới cả hai đỉnh A và B, nhưng theo cách chúng ta sẽ chiếu sáng địa hình bằng kỹ thuật chiếu sáng theo chiều cao, đỉnh A sẽ sáng và đỉnh B sẽ tối, điều này là không chính xác (như hình minh họa).
    - Vấn đề thứ hai với kỹ thuật này là nó cung cấp cho bạn rất ít tự do về cách bạn muốn chiếu sáng địa hình của mình Nhìn. Bây giờ chúng ta cần tiến hành và thảo luận linh hoạt và thực tế hơn cách chiếu sáng địa hình.
  1. Hardware Lighting
     + Kỹ thuật này có hai vấn đề lớn. Đầu tiên, nó phụ thuộc nhiều vào API, vì vậy tôi sẽ không hiển thị mã cho bạn hoặc cung cấp cho bạn bản demo của nó. Thứ hai, nó khá vô dụng đối với các lưới địa hình động - loại mà chúng tôi sẽ làm việc với ba chương tiếp theo. Vì những điều này vấn đề, tôi sẽ chỉ cung cấp cho bạn một số chi tiết triển khai cơ bản tại đây.
     + Ánh sáng phần cứng yêu cầu bạn tính toán bề mặt bình thường cho mọi hình tam giác mà bạn kết xuất. Thời điểm tốt nhất để làm điều này là trong phân đoạn tiền xử lý bản trình diễn của bạn; theo cách đó, các tính toán không làm hỏng chương trình. Sau khi bạn tính toán bình thường, bạn chỉ cần gửi nó vào API cho hình tam giác hiện tại mà bạn muốn hiển thị và bạn đã hoàn tất.
     + THẬN TRỌNG: Đảm bảo rằng bạn đã thiết lập ánh sáng phần cứng chính xác với API trước khi bạn làm bất cứ điều gì. Ánh sáng phần cứng đôi khi có thể là một thiết lập thực sự khó khăn, vì vậy đừng quá ngạc nhiên nếu địa hình của bạn không được chiếu sáng hoặc sáng không chính xác trong lần đầu tiên bạn thử triển khai. Bạn cần phải đảm bảo rằng bạn có nguồn sáng tùy chỉnh (với đúng độ suy giảm, giá trị khuếch tán/đặc trưng/môi trường xung quanh, v.v.). Sau bạn cài đặt đèn của bạn, đảm bảo rằng bạn đã bật nguồn sáng và thành phần ánh sáng của API của bạn nói chung. Nhiều người đến cho tôi các câu hỏi về ánh sáng phần cứng và 75 phần trăm trong số đó đã quên kích hoạt nguồn sáng của họ! Đừng là một thống kê.
     + Kỹ thuật này hoạt động tốt cho các lưới địa hình tĩnh như loại chúng tôi đã sử dụng trong hai chương trước và chương chúng tôi đang sử dụng trong chương này. Nó làm cho ánh sáng động và mô phỏng ngày/đêm trở thành một gió nhẹ. Tuy nhiên, vì ánh sáng phần cứng chủ yếu dựa trên đỉnh, lưới địa hình động trông không đẹp khi phần cứng được thắp sáng. (Năng động các mắt lưới có các đỉnh dịch chuyển liên tục.) Đó là phần thảo luận của chúng ta về ánh sáng phần cứng. Hy vọng bạn không chớp mắt.
  2. Lightmapping
     + Chúng tôi sẽ liên tục sử dụng bản đồ ánh sáng trong suốt cuốn sách này. một bản đồ ánh sáng hoàn toàn giống như một sơ đồ chiều cao (được thảo luận trong Chương 2), ngoại trừ việc thay vì có thông tin về độ cao, bản đồ ánh sáng chỉ chứa thông tin chiếu sáng. Tất cả mã để tải, lưu và hủy bản đồ ánh sáng đều giống với quy trình tương ứng đối với các bản đồ chiều cao (ngoại trừ các chức năng thao tác trên bản đồ ánh sáng xử lý với các biến khác với hàm sơ đồ chiều cao), vì vậy tôi sẽ không lãng phí thời gian quý báu của bạn bằng cách xem lại từng chức năng. Của chúng tôi thông tin bản đồ ánh sáng sẽ được lưu trữ trong kết cấu RAW thang độ xám, giống như bản đồ chiều cao của chúng tôi, ngoại trừ thông tin chỉ trong bản đồ ánh sáng liên quan đến chiếu sáng. Ví dụ, nhìn vào sơ đồ chiều cao và lightmap trong Hình 4.4, rồi xem kết quả đạt được trong hình 4.5. Xem bản đồ ánh sáng ảnh hưởng đến ánh sáng địa hình như thế nào?
     + Xem cách ánh sáng trong Hình 4.5 có dạng hình cầu giống hệt như sơ đồ ánh sáng trong Hình 4.4? Đó là lý do tại sao chúng tôi sử dụng bản đồ ánh sáng: để xác định chính xác loại ánh sáng mà chúng tôi muốn cho một mảng địa hình. Và bởi vì bạn có thể sơ đồ ánh sáng, bạn có thể sử dụng các thuật toán khác nhau để tạo chúng. Bạn có thể tạo lightmap theo nhiều cách. Một số trong những cách này là kỹ thuật chiếu sáng toàn cầu phức tạp nhưng đẹp mắt. Tôi sẽ sẽ chỉ cho bạn một cách như vậy để tạo sơ đồ ánh sáng trong phần tiếp theo.



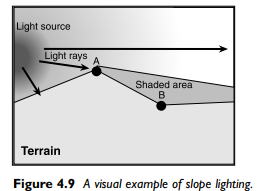
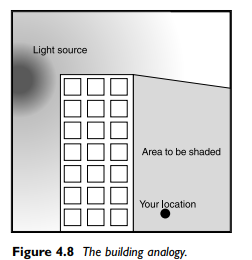
* + - Sau khi bạn đã tải một bản đồ ánh sáng giống như cách chúng tôi đã tải sơ đồ chiều cao, bạn cần tạo một hàm sẽ trích xuất độ sáng tại một pixel nhất định:



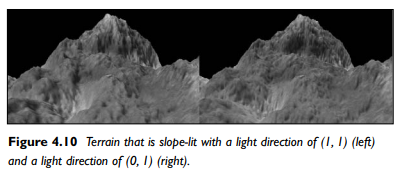
* + - Hãy nhớ cách chúng tôi sử dụng GetTrueHeightAtPoint để lấy độ sáng thông tin về ánh sáng dựa trên chiều cao trong demo4\_2? Tất cả chúng ta phải làm thay cuộc gọi đó bằng một cuộc gọi tới GetBrightnessAtPoint và chúng ta đã sẵn sàng! Nhìn thấy tất cả các kỹ thuật chiếu sáng này dễ dàng như thế nào? Kiểm tra demo4\_2 trên đĩa CD trong Code\Chapter 4\demo4\_2 và thử tạo một số sở hữu những bản đồ chiều cao nhỏ và xem chúng xuất hiện như thế nào. Tôi đã tạo một sơ đồ ánh sáng nhỏ thú vị trong Hình 4.6, với kết quả được hiển thị trong Hình 4.7
    - Như thể điều đó không rõ ràng trước đây, tôi đang phải chịu đựng sự kết hợp cực đoan giữa việc có quá nhiều thời gian và có quá nhiều thời gian. nhiều niềm vui với cuốn sách này!
    - Bản đồ ánh sáng có tầm quan trọng cực kỳ quan trọng trong các trò chơi mà tôi cảm thấy mong muốn mở rộng trên một số tính năng nâng cao hơn của nó. Tôi được bảo hiểm mọi thứ bạn cần làm để “dán” bản đồ ánh sáng lên địa hình, nhưng các khái niệm ánh sáng tiên tiến hơn tập trung vào việc tạo ra một bản đồ ánh sáng. (Tôi sẽ chỉ cho bạn một thuật toán như vậy sau.)



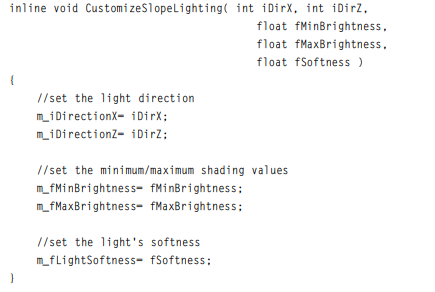
* + - Nhiều trò chơi, chẳng hạn như Max-Payne và Quake 2, sử dụng một phương pháp gọi là phóng xạ. (Truy cập <http://freespace.virgin.net/hugo.elias/radiosity/>radiosity.htm nếu bạn muốn giải thích về kỹ thuật này.) Lưu ý rằng các trò chơi tôi đã đề cập đều là trò chơi trong nhà và địa hình là một chủ đề ngoài trời, như bạn có thể đoán, có nghĩa là chúng tôi phải tìm một kỹ thuật khác để tính toán các bản đồ ánh sáng của chúng tôi. May mắn cho chúng tôi, nhiều kỹ thuật khác nhau có sẵn (gần như là một số tiền trên thực tế). Tôi sẽ cung cấp một thuật toán đơn giản như vậy, nhưng hãy biết rằng đó là một thuật toán đơn giản và gần như không mạnh bằng một số kỹ thuật chiếu sáng toàn cầu khác đang tồn tại, một trong số đó tôi sẽ đề cập đến ở cuối chương này.
  1. Slope Lighting
     + Hãy để tôi lấy cái này ra khỏi hệ thống của tôi ngay bây giờ: Đây là một trong những thứ thú vị nhất các thuật toán mà tôi đã gặp trong một thời gian dài. Nó cực kỳ đơn giản để sử dụng, và nó cung cấp kết quả sắc nét. Chiếu sáng mái dốc 1 đơn giản kỹ thuật chiếu sáng che các đỉnh theo chiều cao của chúng trong quan hệ với một đỉnh lân cận.
     + Okay, Slope Lighting Is Cool,But How Is It Performed?
       - Để làm dốc địa hình nhẹ, chúng ta sẽ lấy chiều cao từ đỉnh bên cạnh đỉnh hiện tại (hướng sẽ được quyết định bởi hướng của ánh sáng), rồi trừ nó theo chiều cao của đỉnh hiện tại. Về cơ bản, chúng ta đang kiểm tra xem liệu đỉnh kia có đổ bóng lên đỉnh hiện tại hay không. Tôi nghĩ đây là thời điểm hoàn hảo cho một ví dụ thực tế cuộc sống. Giả sử bạn đang đứng trước một tòa nhà lớn đang che khuất mặt trời khỏi tầm nhìn của bạn. Tòa nhà sẽ đổ bóng lên bạn, như trong Hình 4.8.
       - Như bạn có thể thấy trong hình, các tia của nguồn sáng sẽ không chiếu tới bạn. vị trí do tòa nhà lớn cản trở họ. kết quả cuối cùng là rằng bạn sẽ đứng trong một khu vực bóng tối, khiến bạn trông tối hơn cho những người đang nhận được tia sáng. Đây cũng chính là khái niệm mà chúng tôi đang cố gắng đạt được trong chiếu sáng dốc—các đỉnh tạo bóng đang có một tia của nguồn sáng bị chặn bởi đỉnh cao hơn trước họ. Khái niệm này được giải thích rõ hơn trong Hình 4.9.



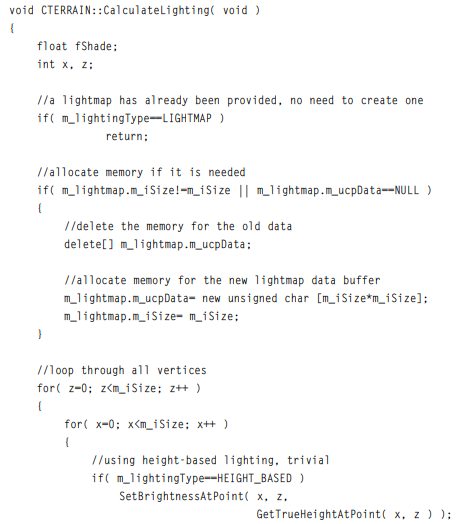
* + - * Như bạn có thể thấy trong hình, nguồn sáng phát ra tất cả các loại tia sáng (thực tế là một lượng gần như vô hạn) nhưng, trong trường hợp này, không có tia nào chúng sẽ đến đỉnh B vì đỉnh A chắn ánh sáng. Chỉ vì đỉnh B không nhận được tia sáng trực tiếp không có nghĩa là nó tối hoàn toàn; một số ánh sáng mà các đỉnh khác nhận được “chảy máu” ra đỉnh, làm sáng nó một chút. một đỉnh là không bao giờ hoàn toàn tối.
      * Có một lỗ hổng nhỏ trong thuật toán này và đó là khi bạn đặt hướng ánh sáng, nó phải tăng dần 45 độ. Ví dụ, phía bên trái của Hình 4.10 được chiếu sáng bởi một ánh sáng có phương (1, 1). Nếu như chúng tôi muốn di chuyển ánh sáng sang trái, chúng tôi sẽ phải thay đổi ánh sáng theo hướng (0, 1), điều này sẽ làm cho ánh sáng di chuyển 45 độ sang trái thay vì chuyển tiếp mượt mà như chuyển động nhẹ từ 2–5 độ một lúc.



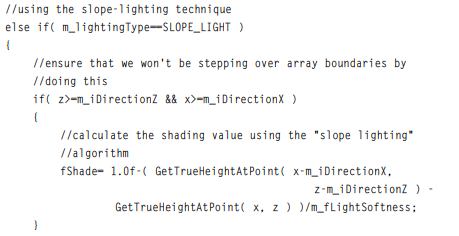
* + - * Bây giờ, nếu bạn để ý, thực sự không có sự khác biệt lớn giữa những bức ảnh này. Do đó, bạn có thể dễ dàng thoát khỏi việc thay đổi hướng ánh sáng trong thời gian thực. Hầu hết người dùng sẽ không nhận thấy "nhảy" nhẹ trong bóng râm.
    - Creating a Slope-Lighting System
      * Chúng ta cần thêm một vài biến vào lớp CTERRAIN trước khi có thể bắt đầu viết mã ánh sáng dốc. (Nếu bạn nghĩ rằng lớp học này đang được lấp đầy với rất nhiều tính năng bổ sung ngay bây giờ, chỉ cần chờ sau này!) Chúng ta cần có thể xác định độ sáng tối thiểu/tối đa cho địa hình của chúng tôi bởi vì, như tôi đã nói trước đó, hiếm khi một đỉnh bị che khuất hoàn toàn tối sầm lại. Chúng ta cũng cần các biến cho độ mềm của ánh sáng và hướng ánh sáng, cả hai đều cần được sử dụng nếu chúng ta muốn đạt được kết quả thực tế. Ngoài ra, chúng tôi muốn một chức năng sẽ dễ dàng cho phép người dùng tùy chỉnh các thông số của hệ thống chiếu sáng mái dốc:



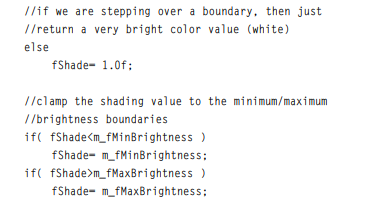
* + - Dynamically Creating Lightmaps
      * Trước khi chúng ta đi xa hơn, bạn có nhớ tôi đã nói rằng điều quan trọng nhất thuật toán lightmapping bao gồm việc tạo lightmap? Chà, bây giờ chúng ta sẽ tạo một chức năng sẽ tạo ra một sơ đồ ánh sáng để sử dụng bởi địa hình của chúng tôi. Bạn có thể chọn tính toán ánh sáng cho mọi khung hình (nó là không phải là một quá trình chậm chạp với các thuật toán mà tôi đang trình bày ở đây), nhưng sẽ tốt hơn nhiều nếu tính toán một lần khi bắt đầu một demo rồi tính toán chiếu sáng lại bất cứ khi nào cần. Đây là nửa đầu của chức năng mà tôi đã tạo để thực hiện việc này:

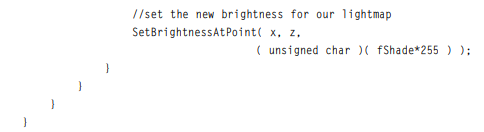


* + - * Cho đến thời điểm này, bạn sẽ có thể hiểu được mọi thứ. Chúng tôi bắt đầu bằng cách kiểm tra xem liệu người dùng có đang sử dụng lightmap. Nếu đúng như vậy, thì chúng tôi không muốn ghi đè thông tin trong bản đồ ánh sáng. Sau đó, chúng ta cần xem liệu chúng ta có cần cấp phát bộ nhớ cho lightmap hay không. Sau đó, chúng tôi bắt đầu lặp qua tất cả các các đỉnh—bản đồ ánh sáng cần có cùng kích thước với bản đồ chiều cao của chúng ta— và kiểm tra xem liệu người dùng có đang sử dụng ánh sáng dựa trên chiều cao hay không. Nếu đúng như vậy, thì chúng tôi đặt pixel hiện tại thành pixel giống như trong sơ đồ chiều cao. Phần còn lại của chức năng liên quan đến chiếu sáng dốc, vì vậy tôi sẽ mô tả nó trong hai phần:

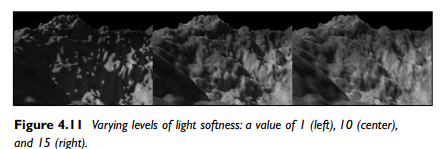


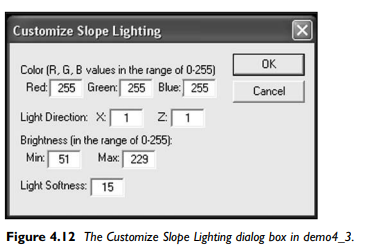
* + - * Đây là nơi diễn ra phần lớn các tính toán chiếu sáng dốc. Như bạn có thể thấy, chúng ta trừ chiều cao của đỉnh trước đỉnh hiện tại một—theo hướng mà người dùng chỉ định—theo đỉnh hiện tại. Chúng tôi đang cố gắng xem bao nhiêu bóng đổ của đỉnh trước đó. Sau đó, chúng tôi chia giá trị đó cho độ mềm của ánh sáng và trừ 1,0f từ giá trị mà chúng tôi có sau khi chia. Tôi thực sự chưa nói về độ dịu của ánh sáng ảnh hưởng đến mọi thứ như thế nào, vì vậy hãy xem Hình 4.11, trong đó Tôi đã chụp ảnh màn hình bằng ba mức độ mềm mại khác nhau.
      * Và bây giờ, phần còn lại của chức năng:





* + - * Trong phần này, chúng tôi kẹp fShade ở độ sáng tối thiểu/tối đa ranh giới và sau đó đặt độ sáng tại điểm hiện tại trong lightmap. Trong demo4\_3 (trên đĩa CD dưới Mã\Chương 4\demo4\_3),Tôi đã thêm một hộp thoại mới thú vị (xem Hình 4.12) cho phép bạn tùy chỉnh hoàn toàn hệ thống chiếu sáng dốc một cách linh hoạt. Có một số niềm vui với nó!





1. Chương 5: