Project 2 - Stereo

- Pyramid construction
- Mutliresolution disparity estimation
  - gray level correlation
  - disparity estimation
  - disparity interpolation
  - disparity map expansion
- Visualization

Pyramid construction

- Given a stereo pair of 256x256 images
- Construct a 3 level pyramid containing images of size 256x256, 128x128, 64x64

Correlation at a given level

- Let D be an estimated disparity image computed from the previous stage of the multiresolution algorithm
  - at first stage, this image is not available, so can be regarded as uniformly 0.
- Correlation algorithm scans through entire left image (ignoring first and last rows and columns) and computes the correlation of the 3x3 neighborhood around $Left_{level}(i,j)$ with the 3x3 neighborhoods in an interval of points around $Right_{level}(i, j + D(i,j))$.

Level 0

- At level 0, the interval considered when matching $Left_{level}(i,j)$ to the right image are the neighborhoods centered around the pixels $Right_{level}(i, j)$ through $Right_{level}(i, j+maxdis/4)$
  - maxdis is the maximum disparity expected to occur in the original full resolution image.
    - Since we have a 3 level pyramid the disparity range at level 0 would be [0,maxdis/4]

Levels 1-2

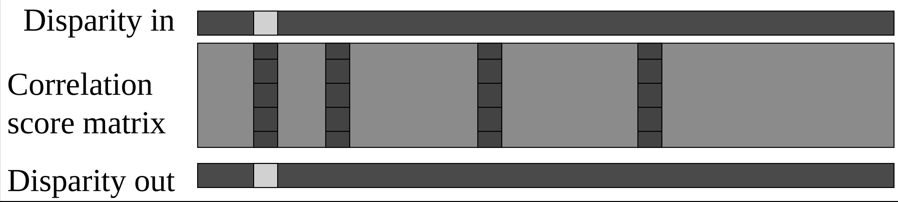
- At levels 1 and 2, we have to match $\text{Left}_{\text{level}}(i,j)$ against an interval "centered" around $\text{Right}_{\text{level}}(i,j+\text{DIS}(i,j))$
  - DIS(i,j) might be slightly inaccurate
  - expansion of pyramid adds a few pixel uncertainty in disparity
  - In any event, do not allow negative disparities

Levels 1-2

- Example: For Left_1 (10,20) our disparity estimate is 10 pixels
  - we "center" our search around the pixel $\text{Right}_1(10,30)$, the predicted match
  - if we allow 3 pixel error in disparity estimate and compute the correlation scores with $[\text{Right}_1(10,27), \text{Right}_1(10,33)]$
    - So, all the correlation scores for a row in the left image can be stored in a 6xCOL matrix, where COL is the number of columns in the image at level i.
    - If any of the 6 entires would arise from a negative disparity, we replace the correlation with maxint.

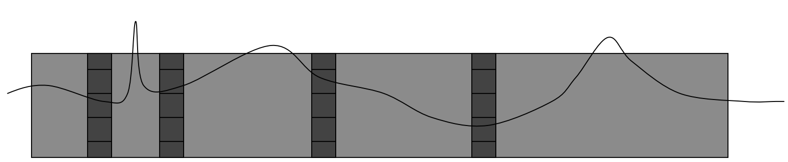
A data structure

- Processing is done one row at a time.
- Goal: Choose a disparity for each column
  - includes a "no disparity" choice for possibly occluded points, with penalty score
  - disparities must satisfy ordering constraint:
    - j+ DIS(j) < (j+1) + DIS(j+1)
  - total correlation score must be minimized



Disparity estimation

- Assign a disparity to each pixel in a row of the left image
  - enforce left-to-right ordering
  - allow for "no-match"
  - solve using dynamic programming



Dynamic programming - when recursion hurts

- Recursive algorithms can sometimes be VERY inefficient
- Fibonacci number

function fib(n)

begin

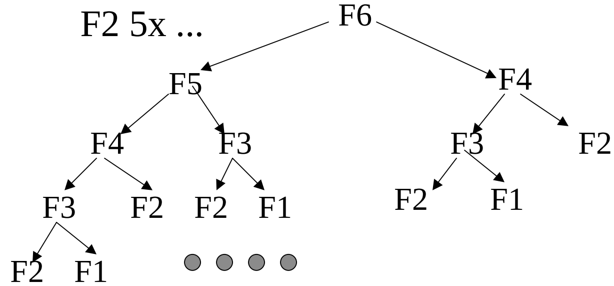
if (n-0) or (n=1) then fib := 1

else fib := fib(n-1) + fib(n-2)

end

Recursive Fibonacci

- For the recursive algorithm T(n) = T(n-1) + T(n-2), which is the same recurrence relation as the sequence itself
  - so, T(n) is exponential
  - F6 is computed once, F5 once, F4 twice, F3 3x, F2 5x ...



Fibonacci

- If the compiler could maintain a table of previously computed Fibonacci numbers, then it could avoid the recursive calls for previously solved subproblems
- This would give us a linear algorithm
- Another time versus space trade-off
  - keep large tables of partial results that must be used over and over to solve a problem
  - only compute each partial result once - when it is first referenced.

A real example - matrix multiplication

- Suppose we have four matrices A (50x10), B(10x40), C(40x30) and D(30x5) and we want to compute ABCD. There are five ways to do this:
  1) A((BC)D) - requiring 16000 multiplication (12000 to compute the 10x30 matrix BC, 1500 more to compute the 10x5 matrix BCD and then 2500 more to compute ABCD)
  2) A(B(CD)) - 10,500
  3) (AB)(CD) - 36,000
  4) (((AB)C)D) - 87,500
  5) (A(BC))D - 34,500

Matrix multiplication

- So, there can be a BIG difference in the amount of work it takes to do the multiplication
- But the number of possible orderings grows quickly with n, the number of matrices

$$T(n) = \sum_{i=1}^{n-1} T(i)T(n-i)$$

- Suppose last multiplication performed is
  - $(A_1A_2\ldots A_i)(A_{i+1}A_{i+2}\ldots A_n)$
  - There are T(i) ways to compute $(A_1A_2\ldots A_i)$
  - There are T(n-i) ways to compute $(A_{i+1}A_{i+2}\ldots A_n)$
  - There are n-1 places we could have cut the problem into two
- Solution is Catalan numbers, which grow exponentially

A dynamic programming solution

- Let c_i be the number of columns in matrix A_i
  - then A_i has c_{i-1} rows
  - A_0 has c_0 rows
  - required for the multiplication to be valid
- Let $m_{L,R}$ be the number of multiplications needed to multiple $A_LA_{L+1}\ldots A_{R-1}A_R$
  - $m_{L,L} = 0$
  - Suppose the LAST multiplication performed is

$$(A_LA_{L+1}\ldots A_i)(A_{i+1}\ldots A_{R-1}A_R)$$

Then the number of multiplications performed is

$$m_{L,i} + m_{i+1,R} + c_{L-1}c_ic_R$$

A dynamic programming solution

- Define $M_{L,R}$ to be the number of multiplications required in an *optimal ordering* of matrices.

$$M_{L,R} = \min_{L \le i \le R}\{M_{L,i} + M_{i+1,R} + c_{L-1}c_i c_R\}$$

- This expression translates directly into a recursive program
  - that would run forever
- But there are only a total of about $n^2/2$ possible values for the $M_{L,R}$ that EVER need to be computed
  - if R-L = k, then the only values needed in the computation of $M_{L,R}$ are $M_{x,y}$ with y-x < k

The program

$A_1 = 3x5, A_2 = 5x8, A_3 = 8x4, A_4 = 4x3$

for L = 1 to n
    $M_{L,L} = 0$;
for k = 1 to n-1   {k is R-L}
    for L = 1 to n-k
    begin
        R = L + k
        $M_{L,R}$ = maxint
        for i = L to R-1
            $M' = M_{Li} + M_{i+1,R} + c_{L-1}c_i c_R$
            if $M' < M_{L,R}$ then $M_{L,R} = M'$

M

R \ L	1	2	3	4
4		256	96	0
3	220	160	0	
2	120	0		
1	0			

First due date

- April 22 - written description of dynamic programming solution you will use in your implementation
  - Must include the optimization formulae and a small hand drawn example showing how it will work.

Disparity map interpolation and expansion

- Double the size of the disparity map by assigning $D_{level}(i,j)$ to $D_{level+1}(2i,2j)$.
- Along each row of $D_{level+1}$ fill in blanks using linear interpolation