**Q1. HDR IMAGING**

**Description**

Dave Coffin의 dcraw 프로그램의 소스코드를 받아 컴파일하여 사용하였다.



사용한 파라미터는 다음과 같다.

-o 1 : 출력물의 색공간을 sRGB로 한다.

-w : white balancing을 수행한다.

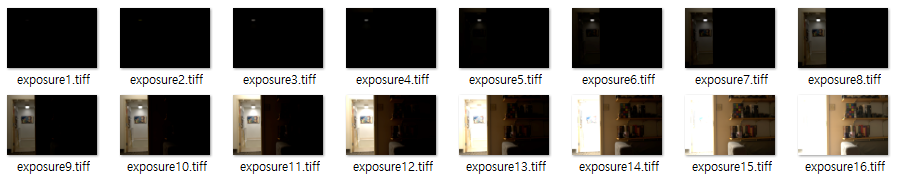
-q 3 : 최대한 퀄리티가 높은 방법으로 interpolation을 수행한다.

-6 : 16bit color로 출력한다.

-W : 이미지를 자동으로 밝게 하지 않는다.

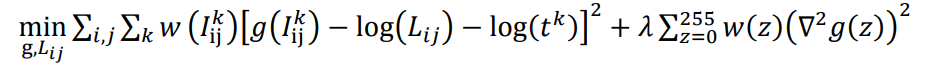
-T : 출력물을 tiff로 한다.

**Result**



**Q2. LINEARIZE RENDERED IMAGES**

**Description**



주어진 공식을 보면 이 공식은 pixel의 위치와는 independent한 식임을 알 수 있다. 따라서 픽셀의 일부를 샘플링하여 실행 속도를 높였다.

가 한 픽셀에서는 상수라는 점에서, 이라는 L과 상관없는 식을 만들 수 있다.

다음으로 에서만 1이고 나머지는 0인 256-D 벡터 를 정의하면, = 로 적을 수 있고 이를 펼치면 하나의 픽셀에 대하여

라는 식을 만들 수 있다. 이 벡터의 1차 미분은 모두 1이다. weight까지 고려하면 하나의 픽셀에 대하여

을 근사적으로 얻을 수 있다.

이제 형태의 식을 얻었으므로 샘플링한 모든 픽셀과 라플라시안 매트릭스까지 더하여 least-squares optimization을 풀면 된다.

이때 모든 등식을 미분 형태로만 구성하였으므로 constant shift가 생길 수 있다. 따라서 g(255) = log(255)로 고정하였다.

사용한 weighting function은 다음과 같다.

|  |  |  |
| --- | --- | --- |
| uniform | tent | gaussian |
|  |  |  |

**Code**

|  |
| --- |
| % Load image as reduced size for reduce memory usage.  image\_origin = {};  for i = 1:16  image\_origin{end+1} = imread(strcat('res\_stack\\exposure',int2str(i),'.tiff'));  image\_origin{end} = im2uint8(imresize(image\_origin{end}, 0.125));  end    for chan = 1:3  % sample 2000 pointe  samples = zeros(32000, 1, 'uint8');  for i = 0:1999  x = randi(750);  y = randi(500);  for f = 1:16  samples(i \* 16 + f) = image\_origin{f}(y, x, chan);  end  end  lambda = 1.0;  % construct G matrix and the differance value b  getg = zeros(32256, 256);  b = zeros(30255, 1);  for i = 0:1999  for j = 1:15  getg((i \* 16 + j), samples((i \* 16 + j)) + 1) = 1;  b(i \* 15 + j) = 0.693147180 \* getw(samples(i \* 16 + j + 1));  end  getg((i \* 16 + 16), samples((i \* 16 + 16)) + 1) = 1;  end    % append original g for create laplacian  for i = 1:256  getg(32000 + i, i) = 1;  end    % set constant (for preserve 255)  b(30255) = log(255);      % set first derivative matrix  diff = zeros(30255, 32256);  for i = 0:1999  for j = 1:15  diff(i \* 15 + j, i \* 16 + j) = -b(i \* 15 + j);  diff(i \* 15 + j, i \* 16 + j + 1) = b(i \* 15 + j);  end  end  % set second derivative (laplacian) matrix  for i = 1:254  diff(30000 + i, 32000 + i) = 1 \* lambda;  diff(30000 + i, 32000 + i + 1) = -2 \* 1 \* lambda;  diff(30000 + i, 32000 + i + 2) = 1 \* lambda;  end  % set constant (for preserve 255)  diff(30255, 32256) = 1;      A = diff \* getg;  g(chan, :) = A \ b;  end |

|  |
| --- |
| function ret = getw(v)  if((v == 0) || (v == 255))  ret = 0;  else  %select weighting function and uncomment one    %ret = exp(-((double(v) - 128.) ^ 2) / 4096);  %ret = (128. - abs(double(v) - 128.)) / 128.;  ret = 1;  end  end |

**Result**

|  |  |  |
| --- | --- | --- |
| w = uniform | w = tent | w = gaussian |
|  |  |  |

**Q2. MERGE EXPOSURE STACK INTO HDR IMAGE & Q3. EVALUATION**

**Description**

linear merging공식으로

logarithmic merging 공식으로

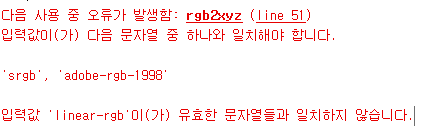
을 사용하였다.

또한 Evaluation을 위해 원본 이미지에 있는



6개 점의 luminacity를 XYZ 색공간에서 비교하여 얼마나 linear한지 확인하는 방법을 사용하였다.

다만 색공간 변환 과정에서



오류가 발생하여 srgb로 비교하였다.

**Code**

|  |
| --- |
| for chan = 1:3  m = 0;  for i = 1:500  for j = 1:750  wsum = 0;  v = 0;  for f = 1:16  % Descripted below  % /--  % --  % --  % --/  wsum = wsum + getw(image\_jpg{f}(i, j, chan));  end  if wsum ~= 0  v = v / wsum;  end  % For Logarithm merging  % /--  v = exp(v);  % --/  image\_res(i, j, chan) = v;  if (m < v)  m = v;  end  end  end  image\_res(:, :, chan) = image\_res(:, :, chan) ./ m;  end  % Image Evaluation  image\_xyz = rgb2xyz(image\_res, 'ColorSpace', 'srgb');  lum = zeros(6, 1);  for f = 1:6  for i = fp(f, 1) - 3 : fp(f, 1) + 3  for j = fp(f, 2) - 3 : fp(f, 2) + 3  lum(f) = lum(f) + image\_xyz(j, i, 2);  end  end  lum(f) = log(lum(f) / 49);  end  lumerr = 0;  for f = 2:5  lumerr = lumerr + ((lum(1) + (lum(6) - lum(1)) \* (f-1) / 5) - lum(f)) ^ 2;  end |

**For Raw, Linear merging**

|  |
| --- |
| v = v + getw(image\_tiff{f}(i, j, chan)) \* ...  double(image\_tiff{f}(i, j, chan)) \* 2 ^ (-f); |

**For Raw Lorarithm merging**

|  |
| --- |
| if(image\_tiff{f}(i, j, chan) > 0)  v = v + getw(image\_tiff{f}(i, j, chan)) \* ...  (log(double(image\_tiff{f}(i, j, chan))) - 0.69314718 \* f);  end |

**For Rendered, Linear merging**

|  |
| --- |
| v = v + getw(image\_jpg{f}(i, j, chan)) \* ...  exp(g(chan, image\_jpg{f}(i, j, chan) + 1)) \* 2 ^ (-f); |

**For Rendered, Lorarithm merging**

|  |
| --- |
| v = v + getw(image\_jpg{f}(i, j, chan)) \* ...  (g(chan, image\_jpg{f}(i, j, chan) + 1) - 0.69314718 \* f); |

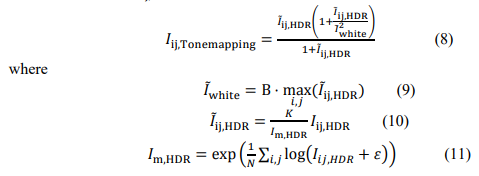
**Result**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | x5 | x125 | x625 |  |
| raw  uniform  linear |  |  |  | 0.03 |
| raw  uniform  logarithm |  |  |  | 0.0167 |
| raw  gaussian  linear |  |  |  | 0.1133 |
| raw  gaussian  logarithm |  |  |  | 0.0325 |
| rendered  uniform  linear |  |  |  | 0.0467 |
| rendered  uniform  logarithm |  |  |  | 0.0053 |
| rendered  gaussian  linear |  |  |  | 0.0510 |
| rendered  gaussian  logarithm |  |  |  | 0.0488 |

**Q4. PHOTOGRAPHIC TONEMAPPING**

**Description**

우선 Tonemapping을 위한 hdr image는 위의 결과중에 가장 나았던 raw 파일, gaussian weight, logarithm merging 이미지를 사용하였다.



공식을 그대로 적용하여 Tonemapping된 픽셀의 밝기를 구했다.

을 사용하였다.

**Code**

|  |
| --- |
| hdr = hdrread('raw\_gaussian\_linear.hdr');    % Constants  K = 0.5;  B = 0.95;    image\_res = zeros(500, 750, 3);    % RGB Tonemappig  for c = 1:3  I\_org = hdr(:, :, c);  I\_m = exp( 1 / (500\*750) \* sum(log(I\_org(:) + 1e-15)) );  I\_tilde = I\_org \* K / I\_m;  I\_white = B \* max(I\_tilde(:));    I\_tone = I\_tilde .\* ( I\_tilde ./ (I\_white ^ 2) + 1) ./ (I\_tilde + 1);  image\_res(:, :, c) = I\_tone(:, :);  end    imwrite(image\_res, 'RGB\_K\_0.5\_B\_0.95.png')    % Get xyY image  hdr\_xyz = rgb2xyz(hdr, 'ColorSpace', 'srgb');  hdr\_xyY = zeros(500, 750, 3);  hdr\_xyY(:, :, 1) = hdr\_xyz(:, : ,1) ./ (hdr\_xyz(:, : ,1) + hdr\_xyz(:, : ,2) + hdr\_xyz(:, : ,3));  hdr\_xyY(:, :, 2) = hdr\_xyz(:, : ,2) ./ (hdr\_xyz(:, : ,1) + hdr\_xyz(:, : ,2) + hdr\_xyz(:, : ,3));  hdr\_xyY(:, :, 3) = hdr\_xyz(:, : ,2);      % luminance Tonemapping  I\_org = hdr\_xyY(:, :, 3);  I\_m = exp( 1 / (500\*750) \* sum(log(I\_org(:) + 1e-15)) );  I\_tilde = I\_org \* K / I\_m;  I\_white = B \* max(I\_tilde(:));    I\_tone = I\_tilde .\* ( I\_tilde ./ (I\_white ^ 2) + 1) ./ (I\_tilde + 1);    % Back to RGB image  image\_res(:, :, 1) = I\_tone(:, :) .\* hdr\_xyY(:, :, 1) ./ hdr\_xyY(:, :, 2);  image\_res(:, :, 2) = I\_tone(:, :);  image\_res(:, :, 3) = I\_tone(:, :) .\* (1 - hdr\_xyY(:, :, 1) - hdr\_xyY(:, :, 2)) ./ hdr\_xyY(:, :, 2);    image\_res = xyz2rgb(image\_res);  imwrite(image\_res, 'Lum\_K\_0.5\_B\_0.95.png') |

**Result**

Change K value ( B = 0.95 )

RGB Tonemapping

|  |  |  |  |
| --- | --- | --- | --- |
| K = 0.05 | K = 0.15 | K = 0.35 | K = 0.5 |
|  |  |  |  |

Luminance Tonemapping

|  |  |  |  |
| --- | --- | --- | --- |
| K = 0.05 | K = 0.15 | K = 0.35 | K = 0.5 |
|  |  |  |  |

Change B value ( K = 0.35 )

RGB Tonemapping

|  |  |  |  |
| --- | --- | --- | --- |
| B = 0.001 | B = 0.01 | B = 0.1 | B = 100 |
|  | C:\Users\KimTaeWoo\AppData\Local\Microsoft\Windows\INetCache\Content.Word\RGB_K_0.35_B_0.01.png | C:\Users\KimTaeWoo\AppData\Local\Microsoft\Windows\INetCache\Content.Word\RGB_K_0.35_B_0.1.png |  |

Luminance Tonemapping

|  |  |  |  |
| --- | --- | --- | --- |
| B = 0.001 | B = 0.01 | B = 0.1 | B = 100 |
|  | C:\Users\KimTaeWoo\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Lum_K_0.35_B_0.01.png | C:\Users\KimTaeWoo\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Lum_K_0.35_B_0.1.png |  |

**Q5. TONEMAPPING USING BILATERAL FILTERING**

우선 Tonemapping을 위한 hdr image는 똑같이 raw 파일, gaussian weight, logarithm merging 이미지를 사용하였다.

Bilateral filter의 적용을 위해 Douglas Lanman의 implement를 사용하였다. 이는 Gaussian inflence function을 사용하며, filter size W와 spartial domain 및 intensity domain 각각의 필터의 표준편차 sigma를 파라미터로 갖는다.

**Code**

|  |
| --- |
| hdr = hdrread('raw\_gaussian\_logarithm.hdr');    % Constants  S = 0.3;  W = 1.;  sigma = [5., 1.0];      image\_res = zeros(500, 750, 3);    % RGB Tonemappig  for c = 1:3  L\_img = log(hdr(:, :, c));  B\_img = bfilter2(L\_img, W, sigma);  D\_img = L\_img - B\_img;  B\_img = (B\_img - max(B\_img(:))) .\* S;    I\_tone = exp(D\_img + B\_img);  image\_res(:, :, c) = I\_tone(:, :);  end    imwrite(image\_res, 'RGB\_S\_0.5\_W\_5\_sigma\_3\_0.5.png')    % Get xyY image  hdr\_xyz = rgb2xyz(hdr, 'ColorSpace', 'srgb');  hdr\_xyY = zeros(500, 750, 3);  hdr\_xyY(:, :, 1) = hdr\_xyz(:, : ,1) ./ (hdr\_xyz(:, : ,1) + hdr\_xyz(:, : ,2) + hdr\_xyz(:, : ,3));  hdr\_xyY(:, :, 2) = hdr\_xyz(:, : ,2) ./ (hdr\_xyz(:, : ,1) + hdr\_xyz(:, : ,2) + hdr\_xyz(:, : ,3));  hdr\_xyY(:, :, 3) = hdr\_xyz(:, : ,2);      % luminance Tonemapping  L\_img = log(hdr\_xyY(:, :, 3));  B\_img = bfilter2(L\_img, W, sigma);  D\_img = L\_img - B\_img;  B\_img = (B\_img - max(B\_img(:))) .\* S;    I\_tone = exp(D\_img + B\_img);    % Back to RGB image  image\_res(:, :, 1) = I\_tone(:, :) .\* hdr\_xyY(:, :, 1) ./ hdr\_xyY(:, :, 2);  image\_res(:, :, 2) = I\_tone(:, :);  image\_res(:, :, 3) = I\_tone(:, :) .\* (1 - hdr\_xyY(:, :, 1) - hdr\_xyY(:, :, 2)) ./ hdr\_xyY(:, :, 2);    image\_res = xyz2rgb(image\_res);  imwrite(image\_res, 'Lum\_S\_0.3\_W\_1\_sigma\_5\_1.png') |

**Result**

Change S value

RGB Tonemapping (W = 5, s\_sprtial = 3, s\_intensity = 0.1)

|  |  |  |  |
| --- | --- | --- | --- |
| S = 0.1 | S = 0.3 | S = 0.5 | S = 1 |
|  |  |  |  |

Luminance Tonemapping (W = 5, s\_sprtial = 3, s\_intensity = 0.1)

|  |  |  |  |
| --- | --- | --- | --- |
| S = 0.1 | S = 0.3 | S = 0.5 | S = 1 |
|  |  |  |  |

Change W value (S = 0.2, s\_sprtial = 5, s\_intensity = 1)

W를 늘리자 연산 시간이 눈에 띄게 증가하였다.

|  |  |  |  |
| --- | --- | --- | --- |
| W = 1 | W = 5 | W = 10 | W = 20 |
| time : 2.1207s | time : 2.6772s | time : 4.9092s | time : 9.5326s |
|  |  |  |  |
|  |  |  |  |

Change s\_intensity value (S = 0.2, W = 5, s\_sprtial = 5)

|  |  |  |  |
| --- | --- | --- | --- |
| s\_intensity = 0.1 | s\_intensity = 0.3 | s\_intensity = 1 | s\_intensity = 5 |
|  |  |  |  |
|  |  |  |  |

Change s\_spartial value (S = 0.2, W = 5, s\_intensity = 1)

|  |  |  |  |
| --- | --- | --- | --- |
| s\_spartial = 0.1 | s\_spartial = 1 | s\_spartial = 5 | s\_spartial = 20 |
|  |  |  |  |
|  |  |  |  |

**Q6. IMPLEMENT A DIFFERENT GRADIENT-DOMAIN PROCESSING ALGORITHM**

기본적인 아이디어는 이미지의 각 영역에 scale factor를 주어 gradient를 더 많이 반영할 영역과 더 적게 반영할 영역을 나누는 것이다.

이후 Poisson Equation을 푸는데,

에서, I는 이미지가 되고 G는 scale이 반영된 Gradient가 된다.

Gradient Attenuation은 기본적으로 의 공식을 적용하여 alpha보다 작은 값은 크게, alpha보다 큰 값은 작게 scailing한다.

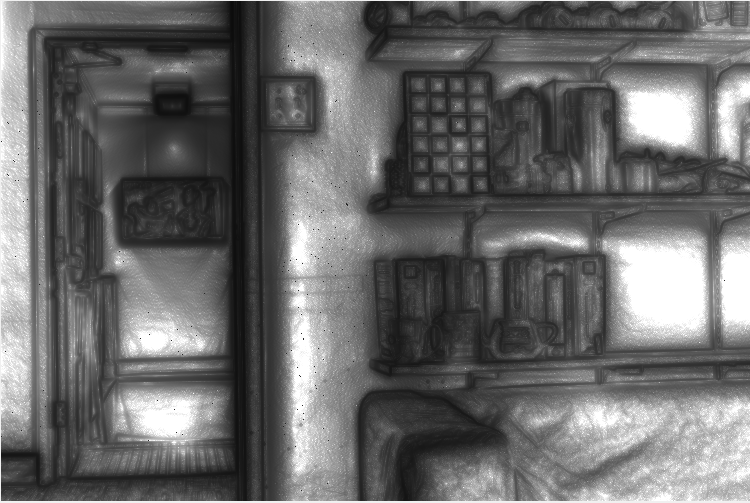
위 함수의 alpha = 0.1, beta = 0.9에서의 그래프는 다음과 같다.



이미지의 Gaussian Pyramid를 구한 후 아래층부터 scaling factor를 구하여 upscaling하고 다음 층의 scaling factor를 곱하는 방식으로 전체 이미지의 scaling factor를 구한다.

이떄 위 공식은 gradient = 0이면 정의되지 않으므로 gradient < 0.0001일 경우 3을 대신 사용하도록 조정하였다.

결과적으로 우리 사진에서 다음 이미지를 얻을 수 있었다.



이제 이미지의 G에 위의 scailing factor를 곱하고, div G를 구해 poisson equation을 풀면 결과가 나온다.

**Code**

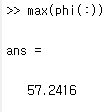
|  |
| --- |
| hdr = hdrread('raw\_gaussian\_logarithm.hdr');    ALPHA = 0.1;  BETA = 0.9;  image\_res = zeros(498, 748, 3);    % Get xyY image  hdr\_xyz = rgb2xyz(hdr, 'ColorSpace', 'srgb');  hdr\_xyY = zeros(500, 750, 3);  hdr\_xyY(:, :, 1) = hdr\_xyz(:, : ,1) ./ (hdr\_xyz(:, : ,1) + hdr\_xyz(:, : ,2) + hdr\_xyz(:, : ,3));  hdr\_xyY(:, :, 2) = hdr\_xyz(:, : ,2) ./ (hdr\_xyz(:, : ,1) + hdr\_xyz(:, : ,2) + hdr\_xyz(:, : ,3));  hdr\_xyY(:, :, 3) = hdr\_xyz(:, : ,2);    % Gradiant Tonemapping  L\_img = log(hdr\_xyY(:, :, 3));    % Make Gaussian Pyramid  L\_Gaussian = {};  L\_Gaussian{1} = L\_img;  [w, h] = size(L\_img);  i = 1;  while((w > 5) && (h > 5))  x = imgaussfilt(L\_Gaussian{i}, 2); % Gaussian Filtering  L\_Gaussian{i + 1} = x(1:2:end, 1:2:end);  [w, h] = size(L\_Gaussian{i+1});  i = i + 1;  end    %Make phi (Gradient scale factor)  phi = zeros(4, 6);  for x = 1:4  for y = 1:6  if x == 1  dif = (L\_Gaussian{8}(x + 1, y) - L\_Gaussian{8}(x, y)) ^ 2;  elseif x == 4  dif = (L\_Gaussian{8}(x, y) - L\_Gaussian{8}(x - 1, y)) ^ 2;  else  dif = (L\_Gaussian{8}(x + 1, y) - L\_Gaussian{8}(x - 1, y)) ^ 2;  end  if y == 1  dif = dif + (L\_Gaussian{8}(x, y + 1) - L\_Gaussian{8}(x, y)) ^ 2;  elseif y == 6  dif = dif + (L\_Gaussian{8}(x, y) - L\_Gaussian{8}(x, y - 1)) ^ 2;  else  dif = dif + (L\_Gaussian{8}(x, y + 1) - L\_Gaussian{8}(x, y - 1)) ^ 2;  end  dif = sqrt(dif) \* (2^(-8));  if dif < 0.000001  phi(x, y) = 3.;  else  phi(x, y) = (dif / ALPHA) ^ (BETA - 1.);  end  end  end  for i = 7:-1:1  phi = imresize(phi, 2, 'bilinear');  [w, h] = size(L\_Gaussian{i});  phi = phi(1:w, 1:h);  for x = 1:w  for y = 1:h  if x == 1  dif = (L\_Gaussian{i}(x + 1, y) - L\_Gaussian{i}(x, y)) ^ 2;  elseif x == w  dif = (L\_Gaussian{i}(x, y) - L\_Gaussian{i}(x - 1, y)) ^ 2;  else  dif = (L\_Gaussian{i}(x + 1, y) - L\_Gaussian{i}(x - 1, y)) ^ 2;  end  if y == 1  dif = dif + (L\_Gaussian{i}(x, y + 1) - L\_Gaussian{i}(x, y)) ^ 2;  elseif y == h  dif = dif + (L\_Gaussian{i}(x, y) - L\_Gaussian{i}(x, y - 1)) ^ 2;  else  dif = dif + (L\_Gaussian{i}(x, y + 1) - L\_Gaussian{i}(x, y - 1)) ^ 2;  end  dif = sqrt(dif) \* (2^(-i));  if dif < 0.000001  phi(x, y) = 3.;  else  phi(x, y) = phi(x, y) \* ((dif / ALPHA) ^ (-0.1));  end  end  end  end    im2var = zeros(498, 748, 'uint32');  im2var(1:498\*748) = 1:498\*748;    % Make G with scale factor  G = zeros(499, 749, 2);  for x = 1:499  for y = 1:749  G(x, y, 1) = phi(x, y) \* (L\_img(x + 1, y) - L\_img(x, y));  G(x, y, 2) = phi(x, y) \* (L\_img(x, y + 1) - L\_img(x, y));  end  end    % Make divG  divG = zeros(498\*748 + 1, 1);  for x = 1:498  for y = 1:748  divG(im2var(x,y)) = G(x+1, y+1, 1) - G(x, y+1, 1) + G(x+1, y+1, 2) - G(x+1, y, 2);  end  end  % Constant zero  divG(498\*748 + 1) = 0.;    % Make Laplacian Matrix  W = sparse(498\*748+1, 498\*748);  for x = 1 : 498  for y = 1 : 748  i = 0;  if x ~= 1  W( im2var(x,y), im2var(x-1,y)) = 1;  i = i - 1;  end  if x ~= 498  W( im2var(x,y), im2var(x+1,y)) = 1;  i = i - 1;  end  if y ~= 1  W( im2var(x,y), im2var(x,y-1) ) = 1;  i = i - 1;  end  if y ~= 748  W( im2var(x,y), im2var(x,y+1) ) = 1;  i = i - 1;  end  W( im2var(x,y), im2var(x,y) ) = i;  end  end  % make center point to zero  W(498\*748+1, im2var(250,375)) = 1;    % solve equation  L\_new = W \ divG;    % restore tone value  I\_tone = zeros(498, 748);  I\_tone(:) = exp(L\_new);    % Normalize  I\_tone = (I\_tone - min(I\_tone(:))) / (max(I\_tone(:)) - min(I\_tone(:)));    % Back to RGB image  image\_res(:, :, 1) = I\_tone(:, :) .\* hdr\_xyY(2:499, 2:749, 1) ./ hdr\_xyY(2:499, 2:749, 2);  image\_res(:, :, 2) = I\_tone(:, :);  image\_res(:, :, 3) = I\_tone(:, :) .\* (1 - hdr\_xyY(2:499, 2:749, 1) - hdr\_xyY(2:499, 2:749, 2)) ./ hdr\_xyY(2:499, 2:749, 2);    image\_res = xyz2rgb(image\_res);  imwrite(image\_res, 'Gradient\_Tonemap\_0.1\_0.9.png') |

**Result**

우선 이 코드를 돌린 결과 다음 이미지를 얻었다.



원인은 phi의 값이 너무 컸다.



따라서 phi를 maximum 값으로 나누어 [0, 1]로 Normalize한 후 코드를 돌린 결과 다음 이미지를 얻었다.



이미지가 지나치게 밝아 이번엔 [0, 3]으로 Normalize한 결과는 다음과 같았다.



[0, 5]로 Normalize하면 다음과 같다.



Alpha나 Beta 값은 크게 결과에 영향을 주지 않았다.

|  |  |  |
| --- | --- | --- |
| α = 0.001, β = 0.9 | α = 0.1, β = 0.9 | α = 0.5, β = 0.9 |
|  |  |  |
| α = 0.1, β = 0.5 | α = 0.1, β = 0.9 | α = 0.1, β = 0.99 |
|  |  |  |