3D LUT interpolation

JD Vandenberg
August 28th, 2019

LUT: Look Up Table

- A faster way to implement a function between two discrete finite sets. $A = \{0, 1, 2, 3, \ldots, 2^N - 1\}, f : A \to A$
- A value comes in and the corresponding output value comes out.
- For a 10bit video, the code book has $2^{10} = 1024$ entries for each primary (red, green, blue).

Example:

$$\begin{array}{c|c} 0 & 0 \\ 1 & 0 \\ 2 & 15 \\ 3 & 125 \\ 4 & 14 \\ \vdots & \vdots \\ 1023 & 1020 \end{array}$$

LUT: Look Up Table - 1D

- Such a LUT is referred to as a 1 dimensional LUT (or 1D LUT)
- One 1D LUT can be used to alter the luminance of the image (tone mapping)
- 3x 1D LUT can be used to alter each color primary separately but can't alter a particular color without affecting others (r, g, b)
- A 10bit video signal is composed of a pair of 3 values between 0 and $2^{10} - 1 = 1023$ for each transported pixel.
- Example:
  - $(0, 0, 0)$ - black
  - $(1023, 1023, 1023)$ - white
  - $(1023, 0, 0)$ - red
  - $(0, 1023, 0)$ - green
  - $(0, 0, 1023)$ - blue
  - $(1023, 1023, 0)$ - yellow
  - ...

LUT: Look Up Table - 3D

- A 3D LUT is a LUT containing entries for each possible (r, g, b) triplets.
- Problem: For a 10-bit video signal, this table would have $(2^{10})^3 = 1,073,741,824$ entries each containing a 30-bit value (10 bits per channel) $= 32,212,254,720$ bits $=$ 4.02653184 Gigabyte.
- The system (hardware or software) would have to parse through a 4.03 Gigabyte memory for every pixel.
- In 4K (UHD: $3,840x2,160$) at 30fps, that is $8,294,400$ pixels 30 times per second or $248,832,000$ scans of that Look Up Table per second!

LUT: Look Up Table - 3D

- A 3D LUT is a LUT containing entries for each possible (r, g, b) triplets.
- Problem: For a 10-bit video signal, this table would have $(2^{10})^3 = 1,073,741,824$ entries each containing a 30-bit value (10 bits per channel) $= 32,212,254,720$ bits $=$ 4.02653184 Gigabyte.
- The system (hardware or software) would have to parse through a 4.03 Gigabyte memory for every pixel.
- In 4K (UHD: $3,840x2,160$) at 30fps, that is $8,294,400$ pixels 30 times per second or $248,832,000$ scans of that Look Up Table per second!
- We need to find another way

LUT: Look Up Table

- Solution: storing a sparse 3D LUT and use interpolation to recover values.

LUT: Look Up Table

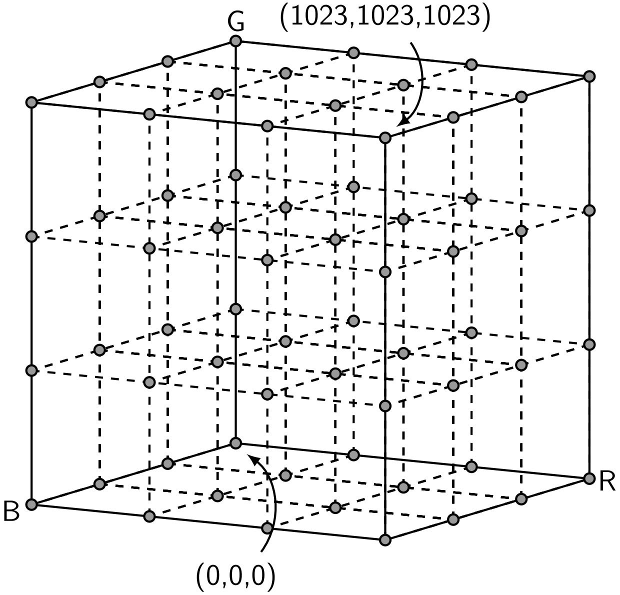
PROS:

- Very light file.
- Typically $33^3 = 35,937$ entries
- intuitive and easy to use.

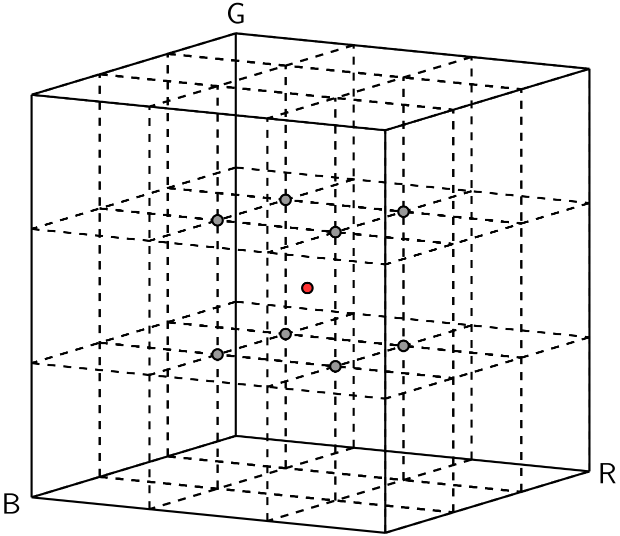
CONS:

- Error due to interpolation
- All the colors are treated the same way
- Hardware often impose table to be be RGB

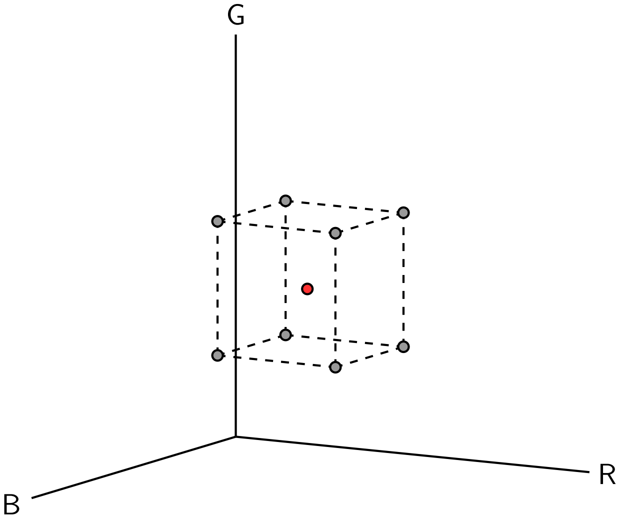
3D LUT interpolation: Trilinear interpolation



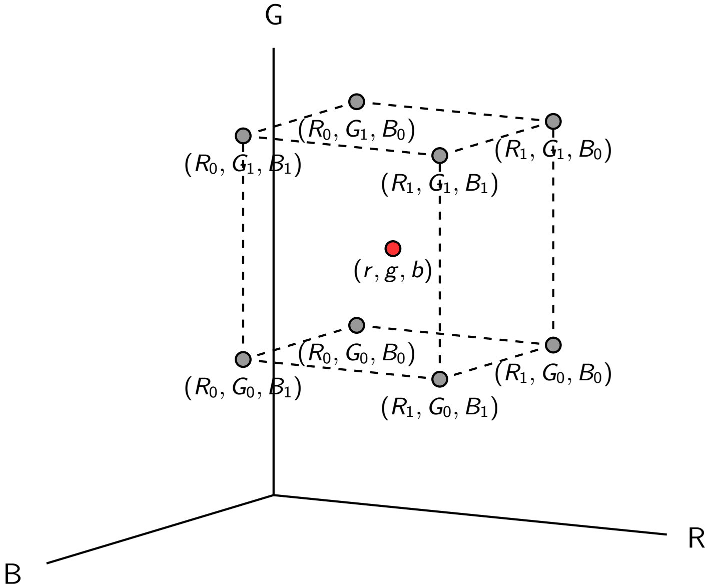
3D LUT interpolation: Trilinear interpolation



3D LUT interpolation: Trilinear interpolation



3D LUT interpolation: Trilinear interpolation



3D LUT interpolation: Trilinear interpolation

Notation: Upper case for mesh points. lower case for points non on the grid.
$V(r, g, b)$ is the value at the point with coordinate (r, g, b).
We start by calculating the distance between each node per coordinate:

$$\Delta_r = \frac{r - R_0}{R_1 - R_0}$$

$$\Delta_g = \frac{g - G_0}{G_1 - G_0}$$

$$\Delta_b = \frac{b - B_0}{B_1 - B_0}$$

This is simply the proportion of each before and after mesh point there is in the point x.

3D LUT interpolation: Trilinear interpolation

We start with red (the result is independent of the order) and calculate the value at each 4 points by doing the weighted average:

$$V(r, G_0, B_0) = V(R_0, G_0, B_0)(1 - \Delta_r) + V(R_1, G_0, B_0)\Delta_r$$

$$V(r, G_0, B_1) = V(R_0, G_0, B_1)(1 - \Delta_r) + V(R_1, G_0, B_1)\Delta_r$$

$$V(r, G_1, B_0) = V(R_0, G_1, B_0)(1 - \Delta_r) + V(R_1, G_1, B_0)\Delta_r$$

$$V(r, G_1, B_1) = V(R_0, G_1, B_1)(1 - \Delta_r) + V(R_1, G_1, B_1)\Delta_r$$

3D LUT interpolation: Trilinear interpolation

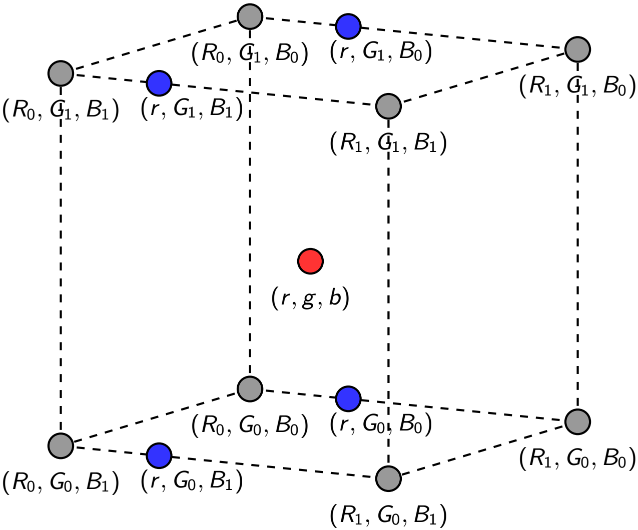
Now that we have computed those values, we move on to the green channel:

$$V(r, g, B_0) = V(r, G_0, B_0)(1 - \Delta_g) + V(r, G_1, B_0)\Delta_g$$
$$V(r, g, B_1) = V(r, G_0, B_1)(1 - \Delta_g) + V(r, G_1, B_1)\Delta_g$$

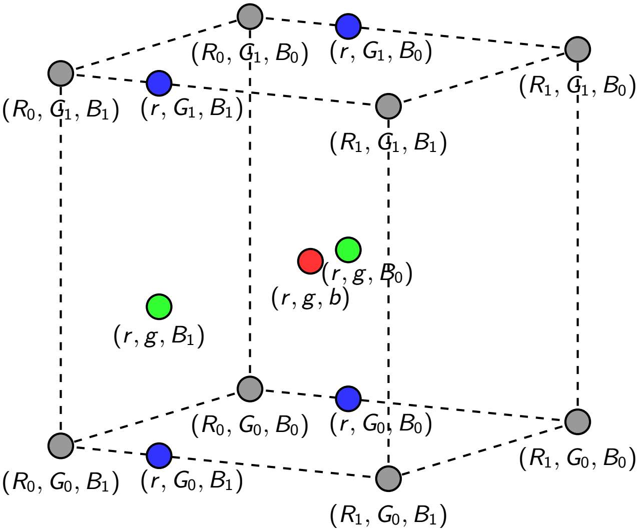
Finally we can compute the value at the point (r, g, b) interpolating the blue channel:

$$V(r, g, b) = V(r, g, B_0)(1 - \Delta_b) + V(r, g, B_1)\Delta_b$$

3D LUT interpolation: Trilinear interpolation



3D LUT interpolation: Trilinear interpolation



3D LUT interpolation: Trilinear interpolation

The general expression for the trilinear interpolation can be expressed as

$$
\begin{aligned}
V(r,g,b) = c_0 + c_1\Delta_b + c_2\Delta_r + c_3\Delta_g + c_4\Delta_b\Delta_r \\
+c_5\Delta_r\Delta_g + c_6\Delta_g\Delta_b + c_7\Delta_r\Delta_g\Delta_b
\end{aligned}
\tag{1}
$$

with:

$c_0 = V(R_0, G_0, B_0)$
$c_1 = V(R_0, G_0, B_1) - V(R_0, G_0, B_0)$
$c_2 = V(R_1, G_0, B_0) - V(R_0, G_0, B_0)$
$c_3 = V(R_0, G_1, B_0) - V(R_0, G_0, B_0)$
$c_4 = V(R_1, G_0, B_1) - V(R_1, G_0, B_0) - V(R_0, G_0, B_1) + V(R_0, G_0, B_0)$
$c_5 = V(R_1, G_1, B_0) - V(R_0, G_1, B_0) - V(R_1, G_0, B_0) + V(R_0, G_0, B_0)$
$c_6 = V(R_0, G_1, B_1) - V(R_0, G_1, B_0) - V(R_0, G_0, B_1) + V(R_0, G_0, B_0)$
$c_7 = V(R_1, G_1, B_1) - V(R_1, G_1, B_0) - V(R_0, G_1, B_1) - V(R_1, G_0, B_1)$
$+V(R_0, G_0, B_1) + V(R_0, G_1, B_0) + V(R_1, G_0, B_0) - V(R_0, G_0, B_0)$

3D LUT interpolation: Trilinear interpolation

Expressed in matrix form:

$$\boldsymbol{C} = \begin{bmatrix} c_0 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 \end{bmatrix}^T$$

$$\boldsymbol{\Delta} = \begin{bmatrix} 1 & \Delta_b & \Delta_r & \Delta_g & \Delta_b\Delta_r & \Delta_r\Delta_g & \Delta_g\Delta_b & \Delta_r\Delta_g\Delta_b \end{bmatrix}^T$$

$$V(r, g, b) = \boldsymbol{C}^T\boldsymbol{\Delta}$$

3D LUT interpolation: Trilinear interpolation

Expressed in matrix form:

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 \\ 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & -1 & 1 & 0 & 0 \\ -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} V(R_0, G_0, B_0) \\ V(R_0, G_1, B_0) \\ V(R_1, G_0, B_0) \\ V(R_1, G_1, B_0) \\ V(R_0, G_0, B_1) \\ V(R_0, G_1, B_1) \\ V(R_1, G_0, B_1) \\ V(R_1, G_1, B_1) \end{bmatrix}$$

The expression above can be written as:
$\boldsymbol{C} = \boldsymbol{AV}$
And the trilinear interpolation as:
$V(r, g, b) = \boldsymbol{C}^T \boldsymbol{\Delta} = \boldsymbol{V}^T \boldsymbol{A}^T \boldsymbol{\Delta}$

3D LUT interpolation: Trilinear interpolation

And the trilinear interpolation can be written as:
$V(r, g, b) = \boldsymbol{C}^T \boldsymbol{\Delta} = \boldsymbol{V}^T \boldsymbol{A}^T \boldsymbol{\Delta}$
Note that the term $\boldsymbol{V}^T \boldsymbol{A}^T$ doesn't depend on the variable (r, g, b) and thus can be computed in advance. So, each sub-cube can have the values of the vector $\boldsymbol{C}$ already stored in memory. Therefore the algorithm can be summarize as:

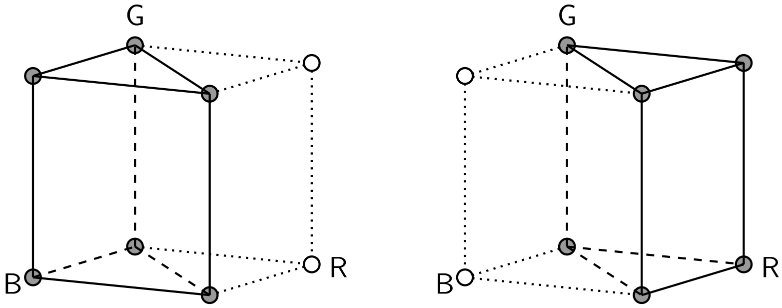
- Find the sub-cube the point (r, g, b) is located in.
- Select the vector $\boldsymbol{C}$ corresponding to that sub-cube.
- Compute Δ_r, Δ_g, Δ_b
- Return $V(r, g, b) = \boldsymbol{C}^T \boldsymbol{\Delta}$

3D LUT interpolation: other interpolations

One idea to speed-up the computation (and hopefully increase its accuracy) is to use a smaller sub-section of the cube.
There are only three ways of slicing a cube into multiple 3D structures with equal number of vertices:

- Prisms (each having six vertices)
- Pyramids (each having five vertices)
- Tetrahedrons (each having four vertices)

3D LUT interpolation: Prism interpolation

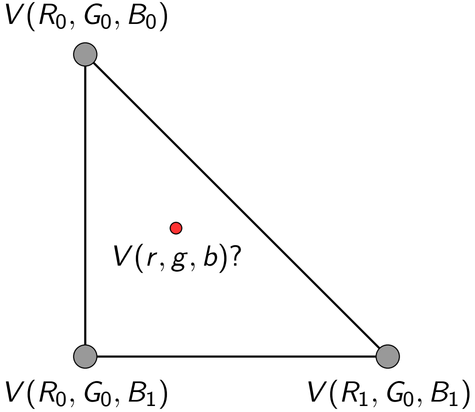


The algorithm chooses the prism based on the cases:

- if $\Delta_b > \Delta_r$, we choose the left prism ($p1$).
- if $\Delta_b < \Delta_r$, we choose the right prism ($p2$).
- if $\Delta_b = \Delta_r$, we choose either prism.

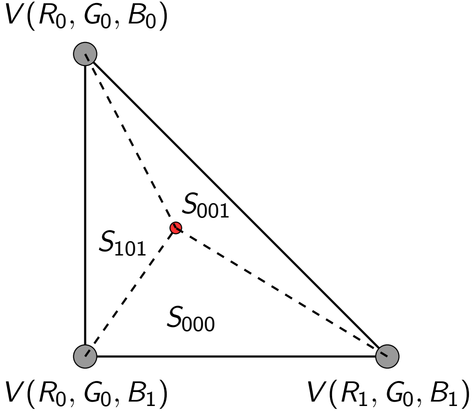
3D LUT interpolation: Prism interpolation

Triangular interpolation:



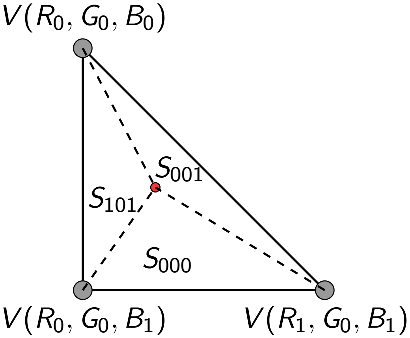
3D LUT interpolation: Prism interpolation

Triangular interpolation:



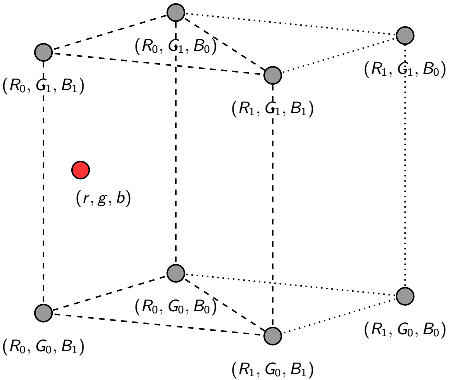
$$V(r, g, b) =$$
$$V(R_0, G_0, B_0)\frac{S_{000}}{S_{tot}} + V(R_0, G_0, B_1)\frac{S_{001}}{S_{tot}} + V(R_1, G_0, B_0)\frac{S_{100}}{S_{tot}}$$

Triangular interpolation:



Surface is given by the shoelace formula:

$$S_{000} = \frac{1}{2}\left| \det \begin{bmatrix} R_0 & R_1 & r \\ B_1 & B_0 & b \\ 1 & 1 & 1 \end{bmatrix} \right| = \frac{1}{2}\left| R_0 B_0 + R1b + rB_1 - R_0 b - R_1 B_1 - rB_0 \right|$$



$$
\begin{aligned}
V(r,g,b) = &\left(V(R_0,G_0,B_0)\frac{S_{000}}{S_{tot}} + V(R_0,G_0,B_1)\frac{S_{001}}{S_{tot}} + V(R_1,G_0,B_1)\frac{S_{101}}{S_{tot}} \right)(1-\Delta_g) \\
&+ \left(V(R_0,G_1,B_0)\frac{S_{010}}{S_{tot}} + V(R_0,G_1,B_1)\frac{S_{011}}{S_{tot}} + V(R_1,G_1,B_1)\frac{S_{111}}{S_{tot}} \right)\Delta_g
\end{aligned} \tag{2}
$$

Note that $S_{000} = S_{010}$, $S_{001} = S_{011}$ and $S_{101} = S_{111}$.

3D LUT interpolation: Prism interpolation

We use the matrix notation and define the solution for both prisms ($p1$, $p2$) using the following vectors:

$$\boldsymbol{V} = \begin{bmatrix} V(R_0,G_0,B_0) \\ V(R_0,G_1,B_0) \\ V(R_1,G_0,B_0) \\ V(R_1,G_1,B_0) \\ V(R_0,G_0,B_1) \\ V(R_0,G_1,B_1) \\ V(R_1,G_0,B_1) \\ V(R_1,G_1,B_1) \end{bmatrix}$$

$$\boldsymbol{\Delta}_p = \begin{bmatrix} 1 & \Delta_b & \Delta_r & \Delta_g & \Delta_b\Delta_g & \Delta_r\Delta_g \end{bmatrix}^T$$

3D LUT interpolation: Prism interpolation

And by defining the two following matrices:

$$
\boldsymbol{B_1} = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\
-1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & -1 & 0 & 0 & -1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & -1 & -1 & 1
\end{bmatrix}
$$

$$
\boldsymbol{B_2} = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 \\
-1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
-1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & -1 & 0 & 0 & -1 & 1 \\
1 & -1 & -1 & 1 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

3D LUT interpolation: Prism interpolation

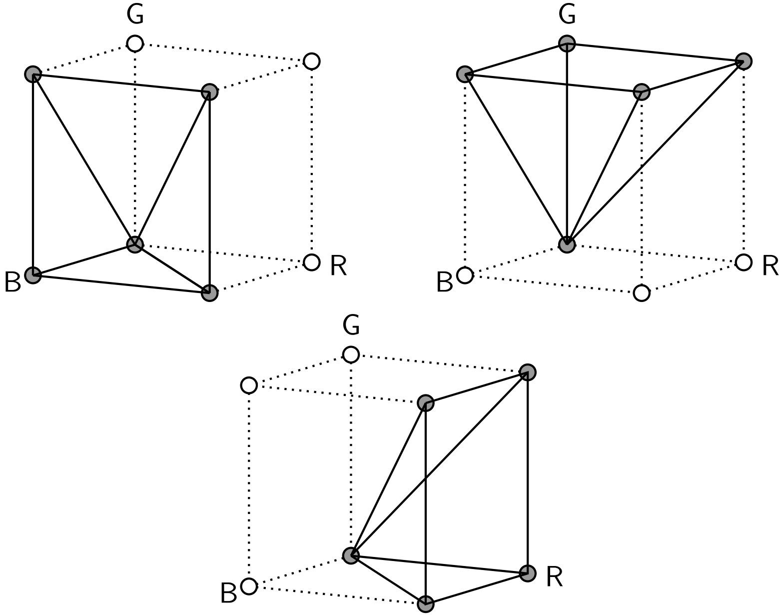
Now we can find the interpolation for each prism as

$$V(r,g,b)_{p_1} = \boldsymbol{\Delta}_p^T \boldsymbol{B_1} \boldsymbol{V}$$

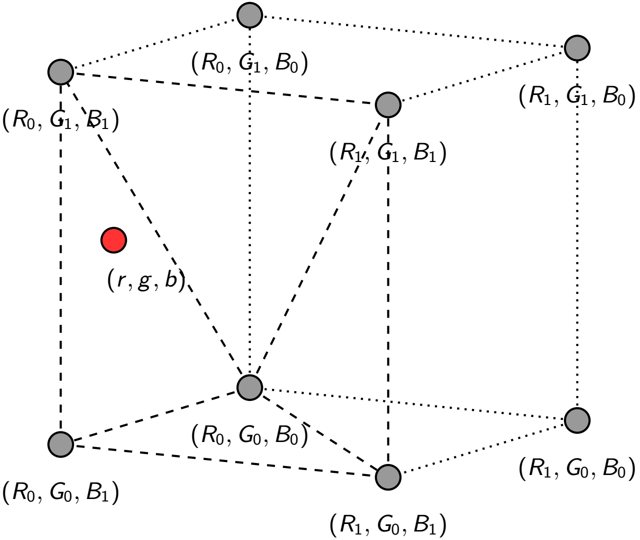
$$V(r,g,b)_{p_2} = \boldsymbol{\Delta}_p^T \boldsymbol{B_2} \boldsymbol{V}$$

We have the same situation as with the trilinear interpolation as the matrices $\boldsymbol{B_1}\boldsymbol{V}$ and $\boldsymbol{B_2}\boldsymbol{V}$ can be computed and stored in advance since they don't depend on the coordinates (r,g,b) or Δ_r, Δ_g and Δ_b. That strategy could speed up the interpolation process.

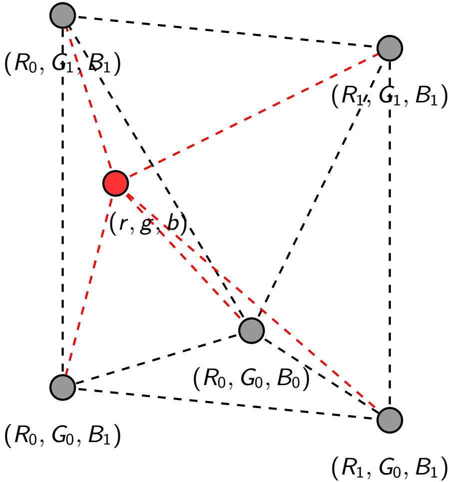
3D LUT interpolation: Pyramid interpolation

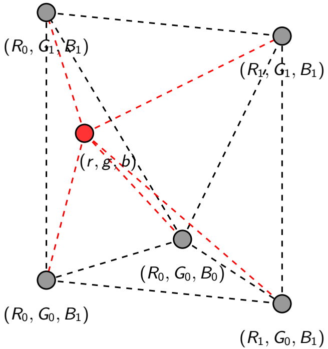


3D LUT interpolation: Pyramid interpolation



3D LUT interpolation: Pyramid interpolation



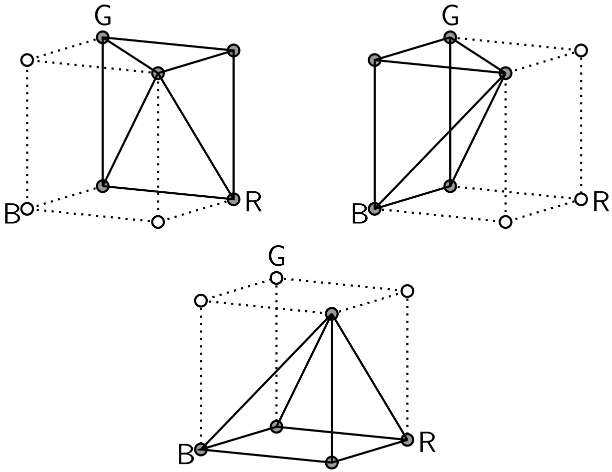


$$
\begin{aligned}
V(r,g,b) = V(R_0, G_0, B_0)\frac{V_{000}}{V_{Prism1}} &+ V(R_0, G_0, B_1)\frac{V_{001}}{V_{Prism1}} \\
+V(R_1, G_0, B_1)\frac{V_{101}}{V_{Prism1}} &+ V(R_0, G_1, B_1)\frac{V_{011}}{V_{Prism1}} \\
&+V(R_1, G_1, B_1)\frac{V_{111}}{V_{Prism1}}
\end{aligned} \tag{3}
$$

Rmk:

$$V_{Prism1} = V_{000} + V_{001} + V_{101} + V_{011} + V_{111}$$

3D LUT interpolation: Pyramid interpolation



The algorithm chooses the pyramid based on the cases:

- if $\Delta_r > \Delta_b$ and $\Delta_g > \Delta_b$, then $p = p_1$.
- if $\Delta_b > \Delta_r$ and $\Delta_g > \Delta_r$, then $p = p_2$.
- else $p = p_3$.

3D LUT interpolation: Pyramid interpolation

We use the matrix notation and define the solution or the three pyramids (p_1, p_2, p_3):

$$\boldsymbol{V} = \begin{bmatrix} V(R_0,G_0,B_0) \\ V(R_0,G_1,B_0) \\ V(R_1,G_0,B_0) \\ V(R_1,G_1,B_0) \\ V(R_0,G_0,B_1) \\ V(R_0,G_1,B_1) \\ V(R_1,G_0,B_1) \\ V(R_1,G_1,B_1) \end{bmatrix}$$

$$\boldsymbol{\Delta}_{p_1} = \begin{bmatrix} 1 & \Delta_b & \Delta_r & \Delta_g & \Delta_r\Delta_g \end{bmatrix}^T$$

$$\boldsymbol{\Delta}_{p_2} = \begin{bmatrix} 1 & \Delta_b & \Delta_r & \Delta_g & \Delta_g\Delta_b \end{bmatrix}^T$$

$$\boldsymbol{\Delta}_{p_3} = \begin{bmatrix} 1 & \Delta_b & \Delta_r & \Delta_g & \Delta_b\Delta_r \end{bmatrix}^T$$

3D LUT interpolation: Pyramid interpolation

And by defining the three following matrices:

$$
\boldsymbol{C_1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}
$$

$$
\boldsymbol{C_2} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & -1 & 1 & 0 & 0 \end{bmatrix}
$$

$$
\boldsymbol{C_3} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 \end{bmatrix}
$$

3D LUT interpolation: Pyramid interpolation

Now we can find the interpolation for each pyramid as

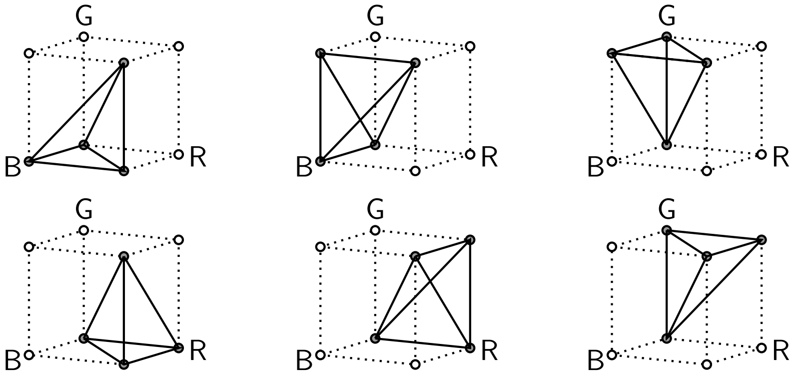
$$V(r,g,b)_{p_1} = \boldsymbol{\Delta}_{p_1}^T \boldsymbol{C_1} \boldsymbol{V}$$

$$V(r,g,b)_{p_2} = \boldsymbol{\Delta}_{p_2}^T \boldsymbol{C_2} \boldsymbol{V}$$

$$V(r,g,b)_{p_3} = \boldsymbol{\Delta}_{p_3}^T \boldsymbol{C_3} \boldsymbol{V}$$

Again, the vectors $\boldsymbol{C_1}\boldsymbol{V}$, $\boldsymbol{C_2}\boldsymbol{V}$ and $\boldsymbol{C_3}\boldsymbol{V}$ can be computed and stored in advance since they don't depend on the coordinates (r,g,b) or Δ_r, Δ_g and Δ_b. That strategy could speed up the interpolation process.

3D LUT interpolation: Tetrahedral interpolation



3D LUT interpolation: Tetrahedral interpolation

The algorithm chooses the tetrahedron based on the cases:

- if $\Delta_b > \Delta_r > \Delta_g$, we chose the first tetrahedron ($t1$),
- if $\Delta_b > \Delta_g > \Delta_r$, we chose the second tetrahedron ($t2$),
- if $\Delta_g > \Delta_b > \Delta_r$, we chose the third tetrahedron ($t3$),
- if $\Delta_r > \Delta_b > \Delta_g$, we chose the fourth tetrahedron ($t4$),
- if $\Delta_r > \Delta_g > \Delta_b$, we chose the fifth tetrahedron ($t5$),
- else we chose the sixth tetrahedron ($t6$).

3D LUT interpolation: Tetrahedral interpolation
We use the matrix notation:

$$\boldsymbol{V} = \begin{bmatrix} V(R_0,G_0,B_0) \\ V(R_0,G_1,B_0) \\ V(R_1,G_0,B_0) \\ V(R_1,G_1,B_0) \\ V(R_0,G_0,B_1) \\ V(R_0,G_1,B_1) \\ V(R_1,G_0,B_1) \\ V(R_1,G_1,B_1) \end{bmatrix}$$

$$\boldsymbol{\Delta}_t = \begin{bmatrix} 1 & \Delta_b & \Delta_r & \Delta_g \end{bmatrix}^T$$

And by defining the six following matrices:

$$T_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}$$

$$T_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \end{bmatrix}$$

$$T_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

3D LUT interpolation: Terahedral interpolation

$$T_4 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}$$

$$T_5 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$T_6 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

3D LUT interpolation: Tetrahedral interpolation

Now we can find the interpolation for each tetrahedron as

$$V(r,g,b)_{t_1} = \boldsymbol{\Delta}_t^T \boldsymbol{T_1 V} \qquad V(r,g,b)_{t_2} = \boldsymbol{\Delta}_t^T \boldsymbol{T_2 V}$$

$$V(r,g,b)_{t_3} = \boldsymbol{\Delta}_t^T \boldsymbol{T_3 V} \qquad V(r,g,b)_{t_4} = \boldsymbol{\Delta}_t^T \boldsymbol{T_4 V}$$

$$V(r,g,b)_{t_5} = \boldsymbol{\Delta}_t^T \boldsymbol{T_5 V} \qquad V(r,g,b)_{t_6} = \boldsymbol{\Delta}_t^T \boldsymbol{T_6 V}$$

Again, the vectors $\boldsymbol{T_i V}$, for $i = 1, 2, 3, 4, 5, 6$, can be computed and stored in advance since they don't depend on the coordinates (r, g, b) or Δ_r, Δ_g and Δ_b. That strategy could speed up the interpolation process.

3D LUT interpolation: comparison of computational cost

	Comparisons	Multiplications	Additions	Storage*
Trilinear	0	7	7	12
Prism	1	5	6	9
Pyramidal	2	4	4	5
Tetrahedron	2.5	3	3	13

* pre-computed coefficient stored at each node

3D LUT interpolation: other approaches

Other ways to interpolate the 3D LUT:

- Cellular Regression
- Non-uniform lattice 3D LUT
- Alternative color spaces

3D LUT interpolation: Cellular Regression

A combination of three-dimensional interpolation and cellular regression.
We apply regression to a small lattice cell (versus the whole cube).

Pros:

- No need to find the position of the interpolation within the cube.
- No need for uniform packing: new 3D structures, like hexahedra, can be used.

Cons:

- Higher computational cost
- Why not use a larger 3D LUT then?

3D LUT interpolation: Non-uniform lattice 3D LUT

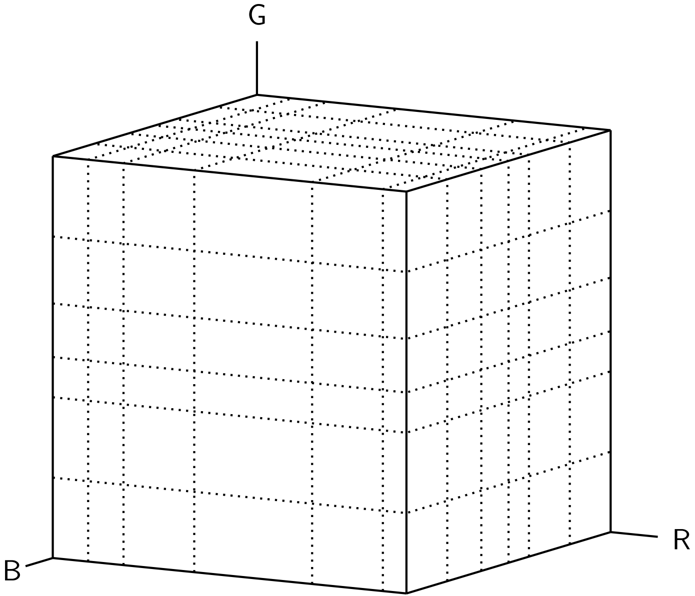
Pros:

- Perception non-uniformity: We don't see difference between two colors the same depending on where we are in the cube.
- Does not require additional computational cost (most hardware feature a front 3x 1D LUT)

Cons:

- Not 'hardware friendly'

3D LUT interpolation: Non-uniform lattice 3D LUT



Acknowledgment

- C. Poynton, Digital Video and HDTV Algorithm and Interfaces, Morgan Kaufmann, San Francisco, 2003.
- Henry R. Kang, Computational Color Technology, SPIE Press Book, 17 May 2006.