SSE

Motivation

- Up to now, we've seen cases where we operate on one data item at a time
  - Or maybe several data items at a time
- Ex: Recall how we manipulated image data...

Image

- Load four RGBA pixels -> XMM register
- Do some operation (brighten, greyscale, etc.)
- Store four RGBA pixels back

Mathematics

- This isn't always feasible, however
- Consider: Geometric calculations
  - I.e., vec3's
- One problem: vec3's don't neatly fit into XMM register
  - We can fit one vec3 in a register, but we waste 25% of computing capacity

Problem

- Second problem: Differing operations
- Suppose we want to do different operations on different RGB color channels
- Can we do this using our current approach?
  - Maybe. Maybe not!

Example

- We want to add +2 to the red channel, +4 to the green, and +1 to the blue
  - Assume it's stored in RAM as r,g,b,a,r,g,b,a,...
  - So if we load to ymm, we get r in slot 0, g in slot 1, etc.
- Load up a YMM:
  ymm[0]=2, ymm[1]=4, ymm[2]=1, ymm[3]=0, ymm[4]=2, ymm[5]=4, etc.
- Do the add. OK!

Example

- But what if we want to do something like:
  - $\frac{1}{4} \cdot r \to$ Shift red right 2
  - $\frac{11}{16} \cdot g \to \frac{1}{2}g + \frac{1}{8}g + \frac{1}{16}g \to$ Shift green right by 1, shift green right by 3, shift green right by 4, and add these
  - $\frac{1}{16} \cdot b \to$ Shift blue right 4
- Hmm. We can't do this
  - AVX/SSE can shift by different amounts...
  - ...But only for 32 bit int's

Storage

- We’ve been using “Array of Structures” scheme
- Ex: We have an array of vec3’s and we want to operate on them
  - vec3 vectors[N];
- This is storage form you’re most familiar with:
- But it’s not the only one we can use

Storage

- “Structure of Arrays” is another way to store data
- We’d store array like so:
  float vectorX[N];
  float vectorY[N];
  float vectorZ[N];

Example

- Suppose we have a bunch of vectors in SoA format and we need to compute their lengths
- We'll use SSE here; AVX uses same ideas

```
alignas(16) float vx[N];  //x coordinates
alignas(16) float vy[N];  //y coordinates
alignas(16) float vz[N];  //z coordinates
alignas(16) float lengths[N];  //output: lengths
```

- Recall: Length of vector = $\sqrt{x^2 + y^2 + z^2}$

Length

- To compute lengths:
- Load multiple elements simultaneously

```
//x holds x coordinates for vectors 0,1,2,3
__m128 x = _mm_load_ps(&vx[0]);
__m128 y = _mm_load_ps(&vy[0]);
__m128 z = _mm_load_ps(&vz[0]);
```

Length

- Compute squared values

```
x = _mm_mul_ps(x,x);
y = _mm_mul_ps(y,y);
z = _mm_mul_ps(z,z);
```

Length

- Compute sum

```
x = _mm_add_ps(x,y);
x = _mm_add_ps(x,z);
```

- Store to memory:
  _mm_store_ps(lengths,x);

SoA

- How does this apply to images?
- Suppose our image data was stored as *bit planes*
  - All the reds, then all the greens, then all the blues
- We could load 32 reds, shift, store
- Then load 32 greens, shift, store
- And finally do the blues

Question

- What if our data isn't stored in planes? What if it's in *chunky pixel* (RGBARGBA...) order?
  - Load 32 pixels' worth of data (8 ymm registers' worth)
  - And then somehow get all the reds in one register, all the greens in another, etc.
- How?
  - Glad you asked!

Problem

- Our usual go-to intrinsics don't seem to be much use here
  - blend can choose between two inputs, but it can't move things around left↔right
  - shuffle can move things left↔right, but it can only work with one input (and only within one lane)
  - The permute family of functions is one of the few that can go cross-lane. But the smallest unit it works with is 32 bit integers
  - We'll only consider:
    c = _mm256_permutevar8x32_epi32(a, b)
    - Gotta love the naming!
    - Treats the 32 byte registers "a," "b," and "c" as sequences of 8 4-byte integers
    - c[i] ← a[b[i]] for i=0…7

Unpack

- c = _mm256_unpacklo_epi8(a , b)
  - c[0] = a[0], c[1] = b[0], c[2] = a[1], c[3] = b[1], ... c[14]=a[7], c[15] = b[7]
  - c[16] = a[16], c[17] = b[16], c[18] = a[17], c[19] = b[17], ... c[31] = b[23]
- This instruction respects the lane boundaries
- In other words, consider the two 16-byte halves of a/b/c separately. Call them a_{lo}, b_{lo}, c_{lo} and a_{hi}, b_{hi}, c_{hi}
  - "Riffle" the low half of a_{lo} (8 bytes) and the low half of b_{lo} (8 bytes) into all of c_{lo} (16 bytes)
  - "Riffle" the low half of a_{hi} (8 bytes) and the low half of b_{hi} (8 bytes) into all of c_{hi} (16 bytes)

Unpack

- c = _mm256_unpackhi_epi8(a , b)
  - "Riffle" the high half of a_{lo} (8 bytes) and the high half of b_{lo} (8 bytes) into all of c_{lo} (16 bytes)
  - "Riffle" the high half of a_{hi} (8 bytes) and the high half of b_{hi} (8 bytes) into all of c_{hi} (16 bytes)

Unpack

- There are unpack's for 8, 16, 32, and 64 bit chunks.
- We will use a progressively larger pack size here...

Step 1

- Load 128 bytes (16 pixels) of data:

```
a=_mm256_lddqu_si256(p);
b=_mm256_lddqu_si256(p+1);
c=_mm256_lddqu_si256(p+2);
d=_mm256_lddqu_si256(p+3);
```

- Result:

$$
\begin{array}{rcccccccccccccccc|cccccccccccccccc}
a = & a_7 & b_7 & g_7 & r_7 & a_6 & b_6 & g_6 & r_6 & a_5 & b_5 & g_5 & r_5 & a_4 & b_4 & g_4 & r_4 & a_3 & b_3 & g_3 & r_3 & a_2 & b_2 & g_2 & r_2 & a_1 & b_1 & g_1 & r_1 & a_0 & b_0 & g_0 & r_0 \\
b = & a_{15} & b_{15} & g_{15} & r_{15} & a_{14} & b_{14} & g_{14} & r_{14} & a_{13} & b_{13} & g_{13} & r_{13} & a_{12} & b_{12} & g_{12} & r_{12} & a_{11} & b_{11} & g_{11} & r_{11} & a_{10} & b_{10} & g_{10} & r_{10} & a_9 & b_9 & g_9 & r_9 & a_8 & b_8 & g_8 & r_8 \\
c = & a_{23} & b_{23} & g_{23} & r_{23} & a_{22} & b_{22} & g_{22} & r_{22} & a_{21} & b_{21} & g_{21} & r_{21} & a_{20} & b_{20} & g_{20} & r_{20} & a_{19} & b_{19} & g_{19} & r_{19} & a_{18} & b_{18} & g_{18} & r_{18} & a_{17} & b_{17} & g_{17} & r_{17} & a_{16} & b_{16} & g_{16} & r_{16} \\
d = & a_{31} & b_{31} & g_{31} & r_{31} & a_{30} & b_{30} & g_{30} & r_{30} & a_{29} & b_{29} & g_{29} & r_{29} & a_{28} & b_{28} & g_{28} & r_{28} & a_{27} & b_{27} & g_{27} & r_{27} & a_{26} & b_{26} & g_{26} & r_{26} & a_{25} & b_{25} & g_{25} & r_{25} & a_{24} & b_{24} & g_{24} & r_{24}
\end{array}
$$

Step 2

```
r1 = _mm256_unpacklo_epi8(a,b)
r2 = _mm256_unpackhi_epi8(a,b)
```

$$
\begin{array}{r|cccccccccccccccc|cccccccccccccccc}
a= & a_7 & b_7 & g_7 & r_7 & a_6 & b_6 & g_6 & r_6 & a_5 & b_5 & g_5 & r_5 & a_4 & b_4 & g_4 & r_4 & a_3 & b_3 & g_3 & r_3 & a_2 & b_2 & g_2 & r_2 & a_1 & b_1 & g_1 & r_1 & a_0 & b_0 & g_0 & r_0 \\
b= & a_{15} & b_{15} & g_{15} & r_{15} & a_{14} & b_{14} & g_{14} & r_{14} & a_{13} & b_{13} & g_{13} & r_{13} & a_{12} & b_{12} & g_{12} & r_{12} & a_{11} & b_{11} & g_{11} & r_{11} & a_{10} & b_{10} & g_{10} & r_{10} & a_9 & b_9 & g_9 & r_9 & a_8 & b_8 & g_8 & r_8 \\
r1= & a_{13} & a_5 & b_{13} & b_5 & g_{13} & g_5 & r_{13} & r_5 & a_{12} & a_4 & b_{12} & b_4 & g_{12} & g_4 & r_{12} & r_4 & a_9 & a_1 & b_9 & b_1 & g_9 & g_1 & r_9 & r_1 & a_8 & a_0 & b_8 & b_0 & g_8 & g_0 & r_8 & r_0 \\
r2= & a_{15} & a_7 & b_{15} & b_7 & g_{15} & g_7 & r_{15} & r_7 & a_{14} & a_6 & b_{14} & b_6 & g_{14} & g_6 & r_{14} & r_6 & a_{11} & a_3 & b_{11} & b_3 & g_{11} & g_3 & r_{11} & r_3 & a_{10} & a_2 & b_{10} & b_2 & g_{10} & g_2 & r_{10} & r_2
\end{array}
$$

Step 3

```
r3 = _mm256_unpacklo_epi16(r1,r2)
r4 = _mm256_unpackhi_epi16(r1,r2)
```

$$
\begin{array}{rcccccccccccccccc|cccccccccccccccc}
r1 = & a_{13} & a_5 & b_{13} & b_5 & g_{13} & g_5 & r_{13} & r_5 & a_{12} & a_4 & b_{12} & b_4 & g_{12} & g_4 & r_{12} & r_4 & a_9 & a_1 & b_9 & b_1 & g_9 & g_1 & r_9 & r_1 & a_8 & a_0 & b_8 & b_0 & g_8 & g_0 & r_8 & r_0 \\
r2 = & a_{15} & a_7 & b_{15} & b_7 & g_{15} & g_7 & r_{15} & r_7 & a_{14} & a_6 & b_{14} & b_6 & g_{14} & g_6 & r_{14} & r_6 & a_{11} & a_3 & b_{11} & b_3 & g_{11} & g_3 & r_{11} & r_3 & a_{10} & a_2 & b_{10} & b_2 & g_{10} & g_2 & r_{10} & r_2 \\
r3 = & a_{14} & a_6 & a_{12} & a_4 & b_{14} & b_6 & b_{12} & b_4 & g_{14} & g_6 & g_{12} & g_4 & r_{14} & r_6 & r_{12} & r_4 & a_{10} & a_2 & a_8 & a_0 & b_{10} & b_2 & b_8 & b_0 & g_{10} & g_2 & g_8 & g_0 & r_{10} & r_2 & r_8 & r_0 \\
r4 = & a_{15} & a_7 & a_{13} & a_5 & b_{15} & b_7 & b_{13} & b_5 & g_{15} & g_7 & g_{13} & g_5 & r_{15} & r_7 & r_{13} & r_5 & a_{11} & a_3 & a_9 & a_1 & b_{11} & b_3 & b_9 & b_1 & g_{11} & g_3 & g_9 & g_1 & r_{11} & r_3 & r_9 & r_1
\end{array}
$$

Step 4

```
r5 = _mm256_unpacklo_epi32(r3,r4)
r6 = _mm256_unpackhi_epi32(r3,r4)
```

$$
\begin{array}{rcccccccccccccccc|cccccccccccccccc}
r3 = & a_{14} & a_{6} & a_{12} & a_{4} & b_{14} & b_{6} & b_{12} & b_{4} & g_{14} & g_{6} & g_{12} & g_{4} & r_{14} & r_{6} & r_{12} & r_{4} & a_{10} & a_{2} & a_{8} & a_{0} & b_{10} & b_{2} & b_{8} & b_{0} & g_{10} & g_{2} & g_{8} & g_{0} & r_{10} & r_{2} & r_{8} & r_{0} \\
r4 = & a_{15} & a_{7} & a_{13} & a_{5} & b_{15} & b_{7} & b_{13} & b_{5} & g_{15} & g_{7} & g_{13} & g_{5} & r_{15} & r_{7} & r_{13} & r_{5} & a_{11} & a_{3} & a_{9} & a_{1} & b_{11} & b_{3} & b_{9} & b_{1} & g_{11} & g_{3} & g_{9} & g_{1} & r_{11} & r_{3} & r_{9} & r_{1} \\
r5 = & g_{15} & g_{7} & g_{13} & g_{5} & g_{14} & g_{6} & g_{12} & g_{4} & r_{15} & r_{7} & r_{13} & r_{5} & r_{14} & r_{6} & r_{12} & r_{4} & g_{11} & g_{3} & g_{9} & g_{1} & g_{10} & g_{2} & g_{8} & g_{0} & r_{11} & r_{3} & r_{9} & r_{1} & r_{10} & r_{2} & r_{8} & r_{0} \\
r6 = & a_{15} & a_{7} & a_{13} & a_{5} & a_{14} & a_{6} & a_{12} & a_{4} & b_{15} & b_{7} & b_{13} & b_{5} & b_{14} & b_{6} & b_{12} & b_{4} & a_{11} & a_{3} & a_{9} & a_{1} & a_{10} & a_{2} & a_{8} & a_{0} & b_{11} & b_{3} & b_{9} & b_{1} & b_{10} & b_{2} & b_{8} & b_{0}
\end{array}
$$

Step 5

```
r7 = _mm256_unpacklo_epi8(c,d)
r8 = _mm256_unpackhi_epi8(c,d)
```

$$
\begin{array}{rcccccccccccccccc|cccccccccccccccc}
c = & a_{23} & b_{23} & g_{23} & r_{23} & a_{22} & b_{22} & g_{22} & r_{22} & a_{21} & b_{21} & g_{21} & r_{21} & a_{20} & b_{20} & g_{20} & r_{20} & a_{19} & b_{19} & g_{19} & r_{19} & a_{18} & b_{18} & g_{18} & r_{18} & a_{17} & b_{17} & g_{17} & r_{17} & a_{16} & b_{16} & g_{16} & r_{16} \\
d = & a_{31} & b_{31} & g_{31} & r_{31} & a_{30} & b_{30} & g_{30} & r_{30} & a_{29} & b_{29} & g_{29} & r_{29} & a_{28} & b_{28} & g_{28} & r_{28} & a_{27} & b_{27} & g_{27} & r_{27} & a_{26} & b_{26} & g_{26} & r_{26} & a_{25} & b_{25} & g_{25} & r_{25} & a_{24} & b_{24} & g_{24} & r_{24} \\
r7 = & a_{29} & a_{21} & b_{29} & b_{21} & g_{29} & g_{21} & r_{29} & r_{21} & a_{28} & a_{20} & b_{28} & b_{20} & g_{28} & g_{20} & r_{28} & r_{20} & a_{25} & a_{17} & b_{25} & b_{17} & g_{25} & g_{17} & r_{25} & r_{17} & a_{24} & a_{16} & b_{24} & b_{16} & g_{24} & g_{16} & r_{24} & r_{16} \\
r8 = & a_{31} & a_{23} & b_{31} & b_{23} & g_{31} & g_{23} & r_{31} & r_{23} & a_{30} & a_{22} & b_{30} & b_{22} & g_{30} & g_{22} & r_{30} & r_{22} & a_{27} & a_{19} & b_{27} & b_{19} & g_{27} & g_{19} & r_{27} & r_{19} & a_{26} & a_{18} & b_{26} & b_{18} & g_{26} & g_{18} & r_{26} & r_{18}
\end{array}
$$

Step 6

```
r9 = _mm256_unpacklo_epi16(r7,r8)
r10 = _mm256_unpackhi_epi16(r7,r8)
```

$r7 =$	a_{29}	a_{21}	b_{29}	b_{21}	g_{29}	g_{21}	r_{29}	r_{21}	a_{28}	a_{20}	b_{28}	b_{20}	g_{28}	g_{20}	r_{28}	r_{20}
$r8 =$	a_{31}	a_{23}	b_{31}	b_{23}	g_{31}	g_{23}	r_{31}	r_{23}	a_{30}	a_{22}	b_{30}	b_{22}	g_{30}	g_{22}	r_{30}	r_{22}
$r9 =$	a_{30}	a_{22}	a_{28}	a_{20}	b_{30}	b_{22}	b_{28}	b_{20}	g_{30}	g_{22}	g_{28}	g_{20}	r_{30}	r_{22}	r_{28}	r_{20}
$r10 =$	a_{31}	a_{23}	a_{29}	a_{21}	b_{31}	b_{23}	b_{29}	b_{21}	g_{31}	g_{23}	g_{29}	g_{21}	r_{31}	r_{23}	r_{29}	r_{21}

$r7 =$	a_{25}	a_{17}	b_{25}	b_{17}	g_{25}	g_{17}	r_{25}	r_{17}	a_{24}	a_{16}	b_{24}	b_{16}	g_{24}	g_{16}	r_{24}	r_{16}
$r8 =$	a_{27}	a_{19}	b_{27}	b_{19}	g_{27}	g_{19}	r_{27}	r_{19}	a_{26}	a_{18}	b_{26}	b_{18}	g_{26}	g_{18}	r_{26}	r_{18}
$r9 =$	a_{26}	a_{18}	a_{24}	a_{16}	b_{26}	b_{18}	b_{24}	b_{16}	g_{26}	g_{18}	g_{24}	g_{16}	r_{26}	r_{18}	r_{24}	r_{16}
$r10 =$	a_{27}	a_{19}	a_{25}	a_{17}	b_{27}	b_{19}	b_{25}	b_{17}	g_{27}	g_{19}	g_{25}	g_{17}	r_{27}	r_{19}	r_{25}	r_{17}

Step 7

```
r11 = _mm256_unpacklo_epi32(r9,r10)
r12 = _mm256_unpackhi_epi32(r9,r10)
```

$$
\begin{array}{r cccccccccccccccc|cccccccccccccccc}
r9 = & a_{30} & a_{22} & a_{28} & a_{20} & b_{30} & b_{22} & b_{28} & b_{20} & g_{30} & g_{22} & g_{28} & g_{20} & r_{30} & r_{22} & r_{28} & r_{20} & a_{26} & a_{18} & a_{24} & a_{16} & b_{26} & b_{18} & b_{24} & b_{16} & g_{26} & g_{18} & g_{24} & g_{16} & r_{26} & r_{18} & r_{24} & r_{16} \\
r10 = & a_{31} & a_{23} & a_{29} & a_{21} & b_{31} & b_{23} & b_{29} & b_{21} & g_{31} & g_{23} & g_{29} & g_{21} & r_{31} & r_{23} & r_{29} & r_{21} & a_{27} & a_{19} & a_{25} & a_{17} & b_{27} & b_{19} & b_{25} & b_{17} & g_{27} & g_{19} & g_{25} & g_{17} & r_{27} & r_{19} & r_{25} & r_{17} \\
r11 = & g_{31} & g_{23} & g_{29} & g_{21} & g_{30} & g_{22} & g_{28} & g_{20} & r_{31} & r_{23} & r_{29} & r_{21} & r_{30} & r_{22} & r_{28} & r_{20} & g_{27} & g_{19} & g_{25} & g_{17} & g_{26} & g_{18} & g_{24} & g_{16} & r_{27} & r_{19} & r_{25} & r_{17} & r_{26} & r_{18} & r_{24} & r_{16} \\
r12 = & a_{31} & a_{23} & a_{29} & a_{21} & a_{30} & a_{22} & a_{28} & a_{20} & b_{31} & b_{23} & b_{29} & b_{21} & b_{30} & b_{22} & b_{28} & b_{20} & a_{27} & a_{19} & a_{25} & a_{17} & a_{26} & a_{18} & a_{24} & a_{16} & b_{27} & b_{19} & b_{25} & b_{17} & b_{26} & b_{18} & b_{24} & b_{16}
\end{array}
$$

Step 8

```
R = _mm256_unpacklo_epi64(r5,r11)
G = _mm256_unpackhi_epi64(r5,r11)
B = _mm256_unpacklo_epi64(r6,r12)
A = _mm256_unpackhi_epi64(r6,r12)
```

$r5 =$	g_{15}	g_{7}	g_{13}	g_{5}	g_{14}	g_{6}	g_{12}	g_{4}	r_{15}	r_{7}	r_{13}	r_{5}	r_{14}	r_{6}	r_{12}	r_{4}	g_{11}	g_{3}	g_{9}	g_{1}	g_{10}	g_{2}	g_{8}	g_{0}	r_{11}	r_{3}	r_{9}	r_{1}	r_{10}	r_{2}	r_{8}	r_{0}
$r11 =$	g_{31}	g_{23}	g_{29}	g_{21}	g_{30}	g_{22}	g_{28}	g_{20}	r_{31}	r_{23}	r_{29}	r_{21}	r_{30}	r_{22}	r_{28}	r_{20}	g_{27}	g_{19}	g_{25}	g_{17}	g_{26}	g_{18}	g_{24}	g_{16}	r_{27}	r_{19}	r_{25}	r_{17}	r_{26}	r_{18}	r_{24}	r_{16}
$r6 =$	a_{15}	a_{7}	a_{13}	a_{5}	a_{14}	a_{6}	a_{12}	a_{4}	b_{15}	b_{7}	b_{13}	b_{5}	b_{14}	b_{6}	b_{12}	b_{4}	a_{11}	a_{3}	a_{9}	a_{1}	a_{10}	a_{2}	a_{8}	a_{0}	b_{11}	b_{3}	b_{9}	b_{1}	b_{10}	b_{2}	b_{8}	b_{0}
$r12 =$	a_{31}	a_{23}	a_{29}	a_{21}	a_{30}	a_{22}	a_{28}	a_{20}	b_{31}	b_{23}	b_{29}	b_{21}	b_{30}	b_{22}	b_{28}	b_{20}	a_{27}	a_{19}	a_{25}	a_{17}	a_{26}	a_{18}	a_{24}	a_{16}	b_{27}	b_{19}	b_{25}	b_{17}	b_{26}	b_{18}	b_{24}	b_{16}
$R =$	r_{31}	r_{23}	r_{29}	r_{21}	r_{30}	r_{22}	r_{28}	r_{20}	r_{15}	r_{7}	r_{13}	r_{5}	r_{14}	r_{6}	r_{12}	r_{4}	r_{27}	r_{19}	r_{25}	r_{17}	r_{26}	r_{18}	r_{24}	r_{16}	r_{11}	r_{3}	r_{9}	r_{1}	r_{10}	r_{2}	r_{8}	r_{0}
$G =$	g_{31}	g_{23}	g_{29}	g_{21}	g_{30}	g_{22}	g_{28}	g_{20}	g_{15}	g_{7}	g_{13}	g_{5}	g_{14}	g_{6}	g_{12}	g_{4}	g_{27}	g_{19}	g_{25}	g_{17}	g_{26}	g_{18}	g_{24}	g_{16}	g_{11}	g_{3}	g_{9}	g_{1}	g_{10}	g_{2}	g_{8}	g_{0}
$B =$	b_{31}	b_{23}	b_{29}	b_{21}	b_{30}	b_{22}	b_{28}	b_{20}	b_{15}	b_{7}	b_{13}	b_{5}	b_{14}	b_{6}	b_{12}	b_{4}	b_{27}	b_{19}	b_{25}	b_{17}	b_{26}	b_{18}	b_{24}	b_{16}	b_{11}	b_{3}	b_{9}	b_{1}	b_{10}	b_{2}	b_{8}	b_{0}
$A =$	a_{31}	a_{23}	a_{29}	a_{21}	a_{30}	a_{22}	a_{28}	a_{20}	a_{15}	a_{7}	a_{13}	a_{5}	a_{14}	a_{6}	a_{12}	a_{4}	a_{27}	a_{19}	a_{25}	a_{17}	a_{26}	a_{18}	a_{24}	a_{16}	a_{11}	a_{3}	a_{9}	a_{1}	a_{10}	a_{2}	a_{8}	a_{0}

Note

- If we need to put the data in order, we can do so with a shuffle (to get 32 bit chunks adjacent) + permute (to put each chunk in the correct spot)
- Do same thing for G, B, A

```
shuffleOrder = _mm256_set_epi8(15,11,13,9, 14,10,12,8, 7,3,5,1,
   6,2,4,0, 15,11,13,9, 14,10,12,8, 7,3,5,1, 6,2,4,0);
rs = _mm_shuffle_epi8( R, shuffleOrder );   //first thing goes to
   slot 31
permuteOrder = _mm256_set_epi32( 7,3,6,2,5,1,4,0 );
rp = _mm256_permutevar8x32_epi32(rs, permuteOrder );
```

$R =$	r_{31}	r_{23}	r_{29}	r_{21}	r_{30}	r_{22}	r_{28}	r_{20}	r_{15}	r_{7}	r_{13}	r_{5}	r_{14}	r_{6}	r_{12}	r_{4}	r_{27}	r_{19}	r_{25}	r_{17}	r_{26}	r_{18}	r_{24}	r_{16}	r_{11}	r_{3}	r_{9}	r_{1}	r_{10}	r_{2}	r_{8}	r_{0}
$rs =$	r_{31}	r_{30}	r_{29}	r_{28}	r_{23}	r_{22}	r_{21}	r_{20}	r_{15}	r_{14}	r_{13}	r_{12}	r_{7}	r_{6}	r_{5}	r_{4}	r_{27}	r_{26}	r_{25}	r_{24}	r_{19}	r_{18}	r_{17}	r_{16}	r_{11}	r_{10}	r_{9}	r_{8}	r_{3}	r_{2}	r_{1}	r_{0}
$rp =$	r_{31}	r_{30}	r_{29}	r_{28}	r_{27}	r_{26}	r_{25}	r_{24}	r_{23}	r_{22}	r_{21}	r_{20}	r_{19}	r_{18}	r_{17}	r_{16}	r_{15}	r_{14}	r_{13}	r_{12}	r_{11}	r_{10}	r_{9}	r_{8}	r_{7}	r_{6}	r_{5}	r_{4}	r_{3}	r_{2}	r_{1}	r_{0}

Result

- For our work here, we don't care if RGBA are in order or out of order
- We need to do these:
  - Approximate red as (red >> 2)
  - Approximate green as (green >> 1) + (green >> 3) + (green >> 4)
  - Approximate blue as (blue >> 4)

Compute

- This is pretty straightforward
  - We saw a right-shift routine last time
  - Red & blue are just a shift
  - Green involves doing shifts and adds
    - AVX has _mm256_add_epi8

Final Operation

- Now we need to reverse the unpacking
- Unfortunately, AVX (and SSE) do *not* have a pack() intrinsic
- But we can still accomplish what we need to do with a bit of ingenuity

Input

- **Our input:**

$$
\begin{array}{l}
R = r_{31}\ r_{23}\ r_{29}\ r_{21}\ r_{30}\ r_{22}\ r_{28}\ r_{20}\ r_{15}\ r_{7}\ r_{13}\ r_{5}\ r_{14}\ r_{6}\ r_{12}\ r_{4} \mid r_{27}\ r_{19}\ r_{25}\ r_{17}\ r_{26}\ r_{18}\ r_{24}\ r_{16}\ r_{11}\ r_{3}\ r_{9}\ r_{1}\ r_{10}\ r_{2}\ r_{8}\ r_{0} \\
G = g_{31}\ g_{23}\ g_{29}\ g_{21}\ g_{30}\ g_{22}\ g_{28}\ g_{20}\ g_{15}\ g_{7}\ g_{13}\ g_{5}\ g_{14}\ g_{6}\ g_{12}\ g_{4} \mid g_{27}\ g_{19}\ g_{25}\ g_{17}\ g_{26}\ g_{18}\ g_{24}\ g_{16}\ g_{11}\ g_{3}\ g_{9}\ g_{1}\ g_{10}\ g_{2}\ g_{8}\ g_{0} \\
B = b_{31}\ b_{23}\ b_{29}\ b_{21}\ b_{30}\ b_{22}\ b_{28}\ b_{20}\ b_{15}\ b_{7}\ b_{13}\ b_{5}\ b_{14}\ b_{6}\ b_{12}\ b_{4} \mid b_{27}\ b_{19}\ b_{25}\ b_{17}\ b_{26}\ b_{18}\ b_{24}\ b_{16}\ b_{11}\ b_{3}\ b_{9}\ b_{1}\ b_{10}\ b_{2}\ b_{8}\ b_{0} \\
A = a_{31}\ a_{23}\ a_{29}\ a_{21}\ a_{30}\ a_{22}\ a_{28}\ a_{20}\ a_{15}\ a_{7}\ a_{13}\ a_{5}\ a_{14}\ a_{6}\ a_{12}\ a_{4} \mid a_{27}\ a_{19}\ a_{25}\ a_{17}\ a_{26}\ a_{18}\ a_{24}\ a_{16}\ a_{11}\ a_{3}\ a_{9}\ a_{1}\ a_{10}\ a_{2}\ a_{8}\ a_{0}
\end{array}
$$

Step 1

```
x1 = _mm256_unpacklo_epi8(R,G);
x2 = _mm256_unpacklo_epi8(B,A);
```

$R =$	r_{31}	r_{23}	r_{29}	r_{21}	r_{30}	r_{22}	r_{28}	r_{20}	r_{15}	r_7	r_{13}	r_5	r_{14}	r_6	r_{12}	r_4	r_{27}	r_{19}	r_{25}	r_{17}	r_{26}	r_{18}	r_{24}	r_{16}	r_{11}	r_3	r_9	r_1	r_{10}	r_2	r_8	r_0
$G =$	g_{31}	g_{23}	g_{29}	g_{21}	g_{30}	g_{22}	g_{28}	g_{20}	g_{15}	g_7	g_{13}	g_5	g_{14}	g_6	g_{12}	g_4	g_{27}	g_{19}	g_{25}	g_{17}	g_{26}	g_{18}	g_{24}	g_{16}	g_{11}	g_3	g_9	g_1	g_{10}	g_2	g_8	g_0
$B =$	b_{31}	b_{23}	b_{29}	b_{21}	b_{30}	b_{22}	b_{28}	b_{20}	b_{15}	b_7	b_{13}	b_5	b_{14}	b_6	b_{12}	b_4	b_{27}	b_{19}	b_{25}	b_{17}	b_{26}	b_{18}	b_{24}	b_{16}	b_{11}	b_3	b_9	b_1	b_{10}	b_2	b_8	b_0
$A =$	a_{31}	a_{23}	a_{29}	a_{21}	a_{30}	a_{22}	a_{28}	a_{20}	a_{15}	a_7	a_{13}	a_5	a_{14}	a_6	a_{12}	a_4	a_{27}	a_{19}	a_{25}	a_{17}	a_{26}	a_{18}	a_{24}	a_{16}	a_{11}	a_3	a_9	a_1	a_{10}	a_2	a_8	a_0
$x1 =$	g_{15}	r_{15}	g_7	r_7	g_{13}	r_{13}	g_5	r_5	g_{14}	r_{14}	g_6	r_6	g_{12}	r_{12}	g_4	r_4	g_{11}	r_{11}	g_3	r_3	g_9	r_9	g_1	r_1	g_{10}	r_{10}	g_2	r_2	g_8	r_8	g_0	r_0
$x2 =$	a_{15}	b_{15}	a_7	b_7	a_{13}	b_{13}	a_5	b_5	a_{14}	b_{14}	a_6	b_6	a_{12}	b_{12}	a_4	b_4	a_{11}	b_{11}	a_3	b_3	a_9	b_9	a_1	b_1	a_{10}	b_{10}	a_2	b_2	a_8	b_8	a_0	b_0

Step 2

```
x3 = _mm256_unpacklo_epi16(x1,x2);
x4 = _mm256_unpackhi_epi16(x1,x2);
```

$$
\begin{array}{r cccccccccccccccc|cccccccccccccccc}
x1 = & g_{15} & r_{15} & g_{7} & r_{7} & g_{13} & r_{13} & g_{5} & r_{5} & g_{14} & r_{14} & g_{6} & r_{6} & g_{12} & r_{12} & g_{4} & r_{4} & g_{11} & r_{11} & g_{3} & r_{3} & g_{9} & r_{9} & g_{1} & r_{1} & g_{10} & r_{10} & g_{2} & r_{2} & g_{8} & r_{8} & g_{0} & r_{0} \\
x2 = & a_{15} & b_{15} & a_{7} & b_{7} & a_{13} & b_{13} & a_{5} & b_{5} & a_{14} & b_{14} & a_{6} & b_{6} & a_{12} & b_{12} & a_{4} & b_{4} & a_{11} & b_{11} & a_{3} & b_{3} & a_{9} & b_{9} & a_{1} & b_{1} & a_{10} & b_{10} & a_{2} & b_{2} & a_{8} & b_{8} & a_{0} & b_{0} \\
x3 = & a_{14} & b_{14} & g_{14} & r_{14} & a_{6} & b_{6} & g_{6} & r_{6} & a_{12} & b_{12} & g_{12} & r_{12} & a_{4} & b_{4} & g_{4} & r_{4} & a_{10} & b_{10} & g_{10} & r_{10} & a_{2} & b_{2} & g_{2} & r_{2} & a_{8} & b_{8} & g_{8} & r_{8} & a_{0} & b_{0} & g_{0} & r_{0} \\
x4 = & a_{15} & b_{15} & g_{15} & r_{15} & a_{7} & b_{7} & g_{7} & r_{7} & a_{13} & b_{13} & g_{13} & r_{13} & a_{5} & b_{5} & g_{5} & r_{5} & a_{11} & b_{11} & g_{11} & r_{11} & a_{3} & b_{3} & g_{3} & r_{3} & a_{9} & b_{9} & g_{9} & r_{9} & a_{1} & b_{1} & g_{1} & r_{1}
\end{array}
$$

Step 3

```
x5 = _mm256_shuffle_epi32(x4, 0xb1 ); //0xb1 = 10110001
out1 = _mm256_blend_epi32(x3,x5, 0xaa); //0xaa = 10101010
```

$$
\begin{array}{rcccccccccccccccc|cccccccccccccccc}
x3 = & a_{14} & b_{14} & g_{14} & r_{14} & a_6 & b_6 & g_6 & r_6 & a_{12} & b_{12} & g_{12} & r_{12} & a_4 & b_4 & g_4 & r_4 & a_{10} & b_{10} & g_{10} & r_{10} & a_2 & b_2 & g_2 & r_2 & a_8 & b_8 & g_8 & r_8 & a_0 & b_0 & g_0 & r_0 \\
x4 = & a_{15} & b_{15} & g_{15} & r_{15} & a_7 & b_7 & g_7 & r_7 & a_{13} & b_{13} & g_{13} & r_{13} & a_5 & b_5 & g_5 & r_5 & a_{11} & b_{11} & g_{11} & r_{11} & a_3 & b_3 & g_3 & r_3 & a_9 & b_9 & g_9 & r_9 & a_1 & b_1 & g_1 & r_1 \\
x5 = & a_7 & b_7 & g_7 & r_7 & a_{15} & b_{15} & g_{15} & r_{15} & a_5 & b_5 & g_5 & r_5 & a_{13} & b_{13} & g_{13} & r_{13} & a_3 & b_3 & g_3 & r_3 & a_{11} & b_{11} & g_{11} & r_{11} & a_1 & b_1 & g_1 & r_1 & a_9 & b_9 & g_9 & r_9 \\
out1 = & a_7 & b_7 & g_7 & r_7 & a_6 & b_6 & g_6 & r_6 & a_5 & b_5 & g_5 & r_5 & a_4 & b_4 & g_4 & r_4 & a_3 & b_3 & g_3 & r_3 & a_2 & b_2 & g_2 & r_2 & a_1 & b_1 & g_1 & r_1 & a_0 & b_0 & g_0 & r_0
\end{array}
$$

Step 4

```
x6 = _mm256_shuffle_epi32(x3, 0xb1);  //0xb1 = 10110001
out2 = _mm256_blend_epi32(x6,x4, 0xaa); //0xaa = 10101010
```

$$\begin{array}{r|cccccccccccccccc|cccccccccccccccc}
x3 = & a_{14} & b_{14} & g_{14} & r_{14} & a_6 & b_6 & g_6 & r_6 & a_{12} & b_{12} & g_{12} & r_{12} & a_4 & b_4 & g_4 & r_4 & a_{10} & b_{10} & g_{10} & r_{10} & a_2 & b_2 & g_2 & r_2 & a_8 & b_8 & g_8 & r_8 & a_0 & b_0 & g_0 & r_0 \\
x4 = & a_{15} & b_{15} & g_{15} & r_{15} & a_7 & b_7 & g_7 & r_7 & a_{13} & b_{13} & g_{13} & r_{13} & a_5 & b_5 & g_5 & r_5 & a_{11} & b_{11} & g_{11} & r_{11} & a_3 & b_3 & g_3 & r_3 & a_9 & b_9 & g_9 & r_9 & a_1 & b_1 & g_1 & r_1 \\
x6 = & a_6 & b_6 & g_6 & r_6 & a_{14} & b_{14} & g_{14} & r_{14} & a_4 & b_4 & g_4 & r_4 & a_{12} & b_{12} & g_{12} & r_{12} & a_2 & b_2 & g_2 & r_2 & a_{10} & b_{10} & g_{10} & r_{10} & a_0 & b_0 & g_0 & r_0 & a_8 & b_8 & g_8 & r_8 \\
out2 = & a_{15} & b_{15} & g_{15} & r_{15} & a_{14} & b_{14} & g_{14} & r_{14} & a_{13} & b_{13} & g_{13} & r_{13} & a_{12} & b_{12} & g_{12} & r_{12} & a_{11} & b_{11} & g_{11} & r_{11} & a_{10} & b_{10} & g_{10} & r_{10} & a_9 & b_9 & g_9 & r_9 & a_8 & b_8 & g_8 & r_8
\end{array}$$

Step 5

```
x7 = _mm256_unpackhi_epi8(R,G);
x8 = _mm256_unpackhi_epi8(B,A);
```

$$
\begin{array}{r cccccccccccccccc|cccccccccccccccc}
R = & r_{31} & r_{23} & r_{29} & r_{21} & r_{30} & r_{22} & r_{28} & r_{20} & r_{15} & r_{7} & r_{13} & r_{5} & r_{14} & r_{6} & r_{12} & r_{4} & r_{27} & r_{19} & r_{25} & r_{17} & r_{26} & r_{18} & r_{24} & r_{16} & r_{11} & r_{3} & r_{9} & r_{1} & r_{10} & r_{2} & r_{8} & r_{0} \\
G = & g_{31} & g_{23} & g_{29} & g_{21} & g_{30} & g_{22} & g_{28} & g_{20} & g_{15} & g_{7} & g_{13} & g_{5} & g_{14} & g_{6} & g_{12} & g_{4} & g_{27} & g_{19} & g_{25} & g_{17} & g_{26} & g_{18} & g_{24} & g_{16} & g_{11} & g_{3} & g_{9} & g_{1} & g_{10} & g_{2} & g_{8} & g_{0} \\
B = & b_{31} & b_{23} & b_{29} & b_{21} & b_{30} & b_{22} & b_{28} & b_{20} & b_{15} & b_{7} & b_{13} & b_{5} & b_{14} & b_{6} & b_{12} & b_{4} & b_{27} & b_{19} & b_{25} & b_{17} & b_{26} & b_{18} & b_{24} & b_{16} & b_{11} & b_{3} & b_{9} & b_{1} & b_{10} & b_{2} & b_{8} & b_{0} \\
A = & a_{31} & a_{23} & a_{29} & a_{21} & a_{30} & a_{22} & a_{28} & a_{20} & a_{15} & a_{7} & a_{13} & a_{5} & a_{14} & a_{6} & a_{12} & a_{4} & a_{27} & a_{19} & a_{25} & a_{17} & a_{26} & a_{18} & a_{24} & a_{16} & a_{11} & a_{3} & a_{9} & a_{1} & a_{10} & a_{2} & a_{8} & a_{0} \\
x7 = & g_{31} & r_{31} & g_{23} & r_{23} & g_{29} & r_{29} & g_{21} & r_{21} & g_{30} & r_{30} & g_{22} & r_{22} & g_{28} & r_{28} & g_{20} & r_{20} & g_{27} & r_{27} & g_{19} & r_{19} & g_{25} & r_{25} & g_{17} & r_{17} & g_{26} & r_{26} & g_{18} & r_{18} & g_{24} & r_{24} & g_{16} & r_{16} \\
x8 = & a_{31} & b_{31} & a_{23} & b_{23} & a_{29} & b_{29} & a_{21} & b_{21} & a_{30} & b_{30} & a_{22} & b_{22} & a_{28} & b_{28} & a_{20} & b_{20} & a_{27} & b_{27} & a_{19} & b_{19} & a_{25} & b_{25} & a_{17} & b_{17} & a_{26} & b_{26} & a_{18} & b_{18} & a_{24} & b_{24} & a_{16} & b_{16}
\end{array}
$$

Step 6

```
x9 = _mm256_unpacklo_epi16(x7,x8);
x10 = _mm256_unpackhi_epi16(x7,x8);
```

$$
\begin{array}{r|cccccccccccccccc|cccccccccccccccc}
x7 = & g_{31} & r_{31} & g_{23} & r_{23} & g_{29} & r_{29} & g_{21} & r_{21} & g_{30} & r_{30} & g_{22} & r_{22} & g_{28} & r_{28} & g_{20} & r_{20} & g_{27} & r_{27} & g_{19} & r_{19} & g_{25} & r_{25} & g_{17} & r_{17} & g_{26} & r_{26} & g_{18} & r_{18} & g_{24} & r_{24} & g_{16} & r_{16} \\
x8 = & a_{31} & b_{31} & a_{23} & b_{23} & a_{29} & b_{29} & a_{21} & b_{21} & a_{30} & b_{30} & a_{22} & b_{22} & a_{28} & b_{28} & a_{20} & b_{20} & a_{27} & b_{27} & a_{19} & b_{19} & a_{25} & b_{25} & a_{17} & b_{17} & a_{26} & b_{26} & a_{18} & b_{18} & a_{24} & b_{24} & a_{16} & b_{16} \\
x9 = & a_{30} & b_{30} & g_{30} & r_{30} & a_{22} & b_{22} & g_{22} & r_{22} & a_{28} & b_{28} & g_{28} & r_{28} & a_{20} & b_{20} & g_{20} & r_{20} & a_{26} & b_{26} & g_{26} & r_{26} & a_{18} & b_{18} & g_{18} & r_{18} & a_{24} & b_{24} & g_{24} & r_{24} & a_{16} & b_{16} & g_{16} & r_{16} \\
x10 = & a_{31} & b_{31} & g_{31} & r_{31} & a_{23} & b_{23} & g_{23} & r_{23} & a_{29} & b_{29} & g_{29} & r_{29} & a_{21} & b_{21} & g_{21} & r_{21} & a_{27} & b_{27} & g_{27} & r_{27} & a_{19} & b_{19} & g_{19} & r_{19} & a_{25} & b_{25} & g_{25} & r_{25} & a_{17} & b_{17} & g_{17} & r_{17}
\end{array}
$$

Step 7

```
x11 = _mm256_shuffle_epi32(x10,0xb1);  //0xb1 = 0b10110001
x12 = _mm256_shuffle_epi32(x9, 0xb1); //0xb1 = 10110001
out3 = _mm256_blend_epi32(x9,x11,0xaa); //0xaa = 0b10101010
out4 = _mm256_blend_epi32(x12,x10,0xaa); //0xaa = 0b10101010
```

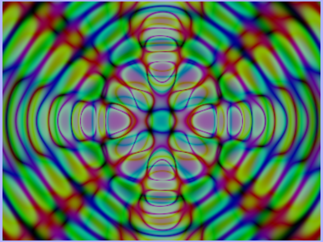
$$\begin{array}{rcccccccccccccccc|cccccccccccccccc}
x11 = & a_{23} & b_{23} & g_{23} & r_{23} & a_{31} & b_{31} & g_{31} & r_{31} & a_{21} & b_{21} & g_{21} & r_{21} & a_{29} & b_{29} & g_{29} & r_{29} & a_{19} & b_{19} & g_{19} & r_{19} & a_{27} & b_{27} & g_{27} & r_{27} & a_{17} & b_{17} & g_{17} & r_{17} & a_{25} & b_{25} & g_{25} & r_{25} \\
x12 = & a_{22} & b_{22} & g_{22} & r_{22} & a_{30} & b_{30} & g_{30} & r_{30} & a_{20} & b_{20} & g_{20} & r_{20} & a_{28} & b_{28} & g_{28} & r_{28} & a_{18} & b_{18} & g_{18} & r_{18} & a_{26} & b_{26} & g_{26} & r_{26} & a_{16} & b_{16} & g_{16} & r_{16} & a_{24} & b_{24} & g_{24} & r_{24} \\
out3 = & a_{23} & b_{23} & g_{23} & r_{23} & a_{22} & b_{22} & g_{22} & r_{22} & a_{21} & b_{21} & g_{21} & r_{21} & a_{20} & b_{20} & g_{20} & r_{20} & a_{19} & b_{19} & g_{19} & r_{19} & a_{18} & b_{18} & g_{18} & r_{18} & a_{17} & b_{17} & g_{17} & r_{17} & a_{16} & b_{16} & g_{16} & r_{16} \\
out4 = & a_{31} & b_{31} & g_{31} & r_{31} & a_{30} & b_{30} & g_{30} & r_{30} & a_{29} & b_{29} & g_{29} & r_{29} & a_{28} & b_{28} & g_{28} & r_{28} & a_{27} & b_{27} & g_{27} & r_{27} & a_{26} & b_{26} & g_{26} & r_{26} & a_{25} & b_{25} & g_{25} & r_{25} & a_{24} & b_{24} & g_{24} & r_{24}
\end{array}$$

Application

- Consider the way special effects are often done
- Blue-screening
- We have two images
  - Live-action, with bluescreen
  - CGI

Application

- Live image, CGI image, and composite:


- How can we accomplish this?

Application

- Suppose input images hold RGBA values (32 bits)
  - Too inconvenient to work with 24 bit (RGB) values
- The string functions won't work here
  - They look at 8 or 16 bit chunks, but we need to look at 24 or 32 bit chunks

Application

- It's unlikely that a live-action shot will have *exactly* 0,0,255 for the bluescreen areas
- Example: Here, we have a source image with "noise" in the blue areas
- Notice: The replacement doesn't look so replacey

Application

- What if we want to match all places where:
  - Red < 50
  - Green < 55
  - Blue > 200
- How to do this?
  - Discuss in class!

Assignment

- Write a program which takes two command line arguments: The name of a live-action PNG file and the name of a CGI file. Do bluescreen replacement (r<50, g<55, b>200) using either SSE or AVX. Write the output to "out.png"
- Example image files are on the class webpage; a non-SIMD example program is here and the Python testbench I used for developing routines is here
- You can assume all the inputs' α values are 255
- Benchmark: AVX=1064 μs, SSE=1168 μs, non-SIMD=1586 μs
  - For reference, memory access time and loop overhead accounted for 840 μs

Sources

- https://stackoverflow.com/questions/6996764/fastest-way-to-do-horizontal-float-vector-sum-on-x86
- John D. Cook. Converting color to grayscale. https://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/
- http://www.equasys.de/colorconversion.html
- https://docs.opencv.org/3.1.0/de/d25/imgproc_color_conversions.html
- Intel Corp. Intel 64 and IA-32 Architecture Optimization Reference Manual.
- https://stackoverflow.com/questions/16425359/should-stdvector-honour-alignofvalue-type

Created using LaTeX.

Main font: Gentium Book Basic, by Victor Gaultney. See http://software.sil.org/gentium/
Monospace font: Source Code Pro, by Paul D. Hunt. See https://fonts.google.com/specimen/Source+Code+Pro and http://sourceforge.net/adobe
Icons by Ulisse Perusin, Steven Garrity, Lapo Calamandrei, Ryan Collier, Rodney Dawes, Andreas Nilsson, Tuomas Kuosmanen, Garrett LeSage, and Jakub Steiner. See http://tango-project.org