Vec3f


```
struct Vec3f
{
  __m128 xyz;
};

void cross(Vec3f& r, const Vec3f& a, const Vec3f& b)
{
  const f128 A = _mm_shuffle_ps(a.xyz, a.xyz, _MM_SHUFFLE(3, 0, 2, 1));
  const f128 B = _mm_shuffle_ps(b.xyz, b.xyz, _MM_SHUFFLE(3, 1, 0, 2));
  const f128 C = _mm_shuffle_ps(a.xyz, a.xyz, _MM_SHUFFLE(3, 1, 0, 2));
  const f128 D = _mm_shuffle_ps(b.xyz, b.xyz, _MM_SHUFFLE(3, 0, 2, 1));
  r.xyz = _mm_fmsub_ps(A, B, _mm_mul_ps(C, D));
}
```

Vec3f Structure of Array (SOA)


```
struct Vec3f
{
  __m256 x;
  __m256 y;
  __m256 z;
};
void cross(Vec3f& __restrict r, const Vec3f& __restrict a,
           const Vec3f& __restrict b)
{
  r.x = _mm256_mul_ps(a.z, b.y);
  r.y = _mm256_mul_ps(a.x, b.z);
  r.z = _mm256_mul_ps(a.y, b.x);
  r.x = _mm256_fmsub_ps(a.y, b.z, r.x);
  r.y = _mm256_fmsub_ps(a.z, b.x, r.y);
  r.z = _mm256_fmsub_ps(a.x, b.y, r.z);
}
```

SIMD-at-Insomniac-Games-How

向量归一化


```
void normalize(Vec3f& v)
{
  __m256 len = _mm256_sqrt_ps(dot(v, v));
  v.x = _mm256_div_ps(v.x, len);
  v.y = _mm256_div_ps(v.y, len);
  v.z = _mm256_div_ps(v.z, len);
}
```

用乘法代替除法


```
void normalize(Vec3f& v)
{
  __m256 len = _mm256_sqrt_ps(dot(v, v));
  len = _mm256_div_ps(_mm256_set1_ps(1.0f), len);
  v.x = _mm256_mul_ps(v.x, len);
  v.y = _mm256_mul_ps(v.y, len);
  v.z = _mm256_mul_ps(v.z, len);
}
```

rcp指令

- 除法仍然是最慢的计算
  - 在Haswell架构上，vdivps的latency是17-21，throughput是13。

- vrcpps近似计算倒数指令
  - 在Haswell架构上，latency是7，throughput是2。

```
void normalize(Vec3f& v)
{
  __m256 len = _mm256_sqrt_ps(dot(v, v));
  len = _mm256_rcp_ps(len);
  v.x = _mm256_mul_ps(v.x, len);
  v.y = _mm256_mul_ps(v.y, len);
  v.z = _mm256_mul_ps(v.z, len);
}
```

完全使用rcp代替div？

rcp误差

- The maximum relative error for this approximation is less than 1.5*2^-12
  - _mm256_rcp_ps

- rcp(1.0f) = 0.999756。
  - normalize()用于模型表面法线归一化
  - 几何相交测试

- _mm256_rsqrt_ps

std::vector


```
void makeNumbers(std::vector<int>& values)
{
  for (int i = 0; i < 128; ++i)
  {
    values.push_back(i);
  }
}

std::vector<int> vv;
makeNumbers(vv);
```

std::vector


```
void makeNumbers(std::vector<int>& values)
{
  values.reserve(128);
  for (int i = 0; i < 128; ++i)
  {
    values.push_back(i);
  }
}

std::vector<int> vv;
makeNumbers(vv);
```

std::vector::push_back()


```
template<typename T>
void vector<T>::push_back(const T&v)
{
  if (size() == capacity()) // 分支阻碍代码向量化
    reserve(capacity() * 1.4);
  new (m_end)T(v);
  m_end++;
}
```

```
std::vector<int> makeNumbers()
{
  int n{0};
  std::vector<int> v(128);
  std::generate(v.begin(), v.end(), [&n] {return (n++);});
  return v; // Name
}

void makeNumbers(std::vector<int>&v)
{
  int n{0};
  v.resize(128);
  std::generate(v.begin(), v.end()[&n] {return(n++);});
}
```

https://godbolt.org/g/n5M2iP

取得std::vector数据地址


```cpp
void func(std::vector<float>& values)
{
    float* pv = &values[0];
    // ...
}

void func(std::vector<float>& values)
{
    float* pv = &values.front();
    // ...
}
```

http://en.cppreference.com/w/cpp/container/vector/data

其他性能优化重要议题

- 数据结构(data structure)和算法
- 缓存优化 O(cache miss)
- SIMD指令解决更复杂问题
  - 数据结构(data layout)
  - branch, gather, scatter
  - Intel Use SIMD Data Layout Templates (SDLT) Efficiently in Animation
- 多线程
  - OpenMP
  - Intel Threading Building Block

小结

- 现代CPU结构被设计得更宽，指令执行更聪明
  - CppCon 2017: Chandler Carruth "Going Nowhere Faster"
- 现代编译器对代码模式的理解更准确，优化程度更高
- 程序员帮助CPU更好地执行编译器产生的更优的代码
  - Stephan T. Lavavej Don't Help the Compiler