C++ Toronto

A Small Talk About Arrays

Mark Elendt | SideFX

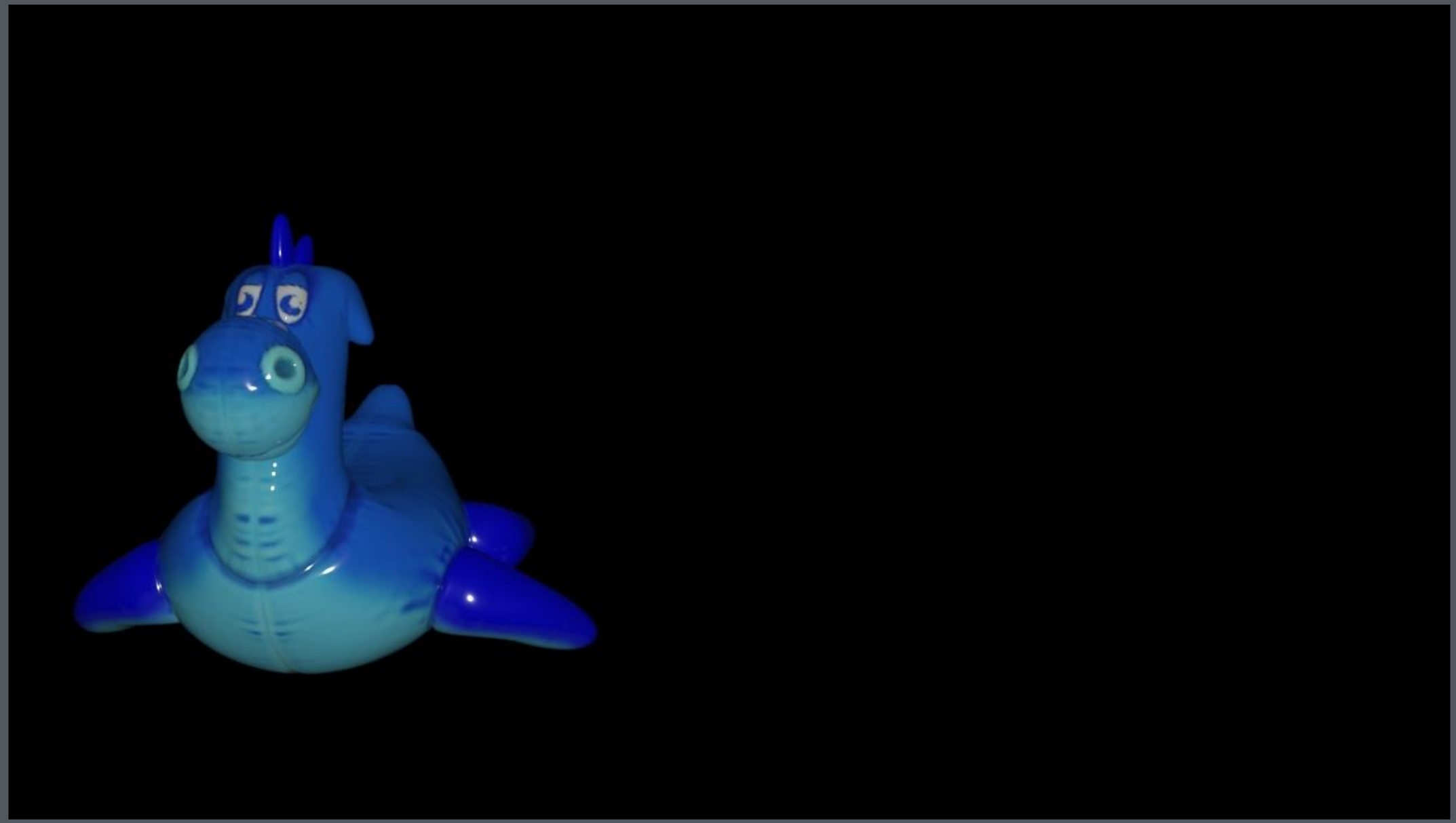
SideFX®

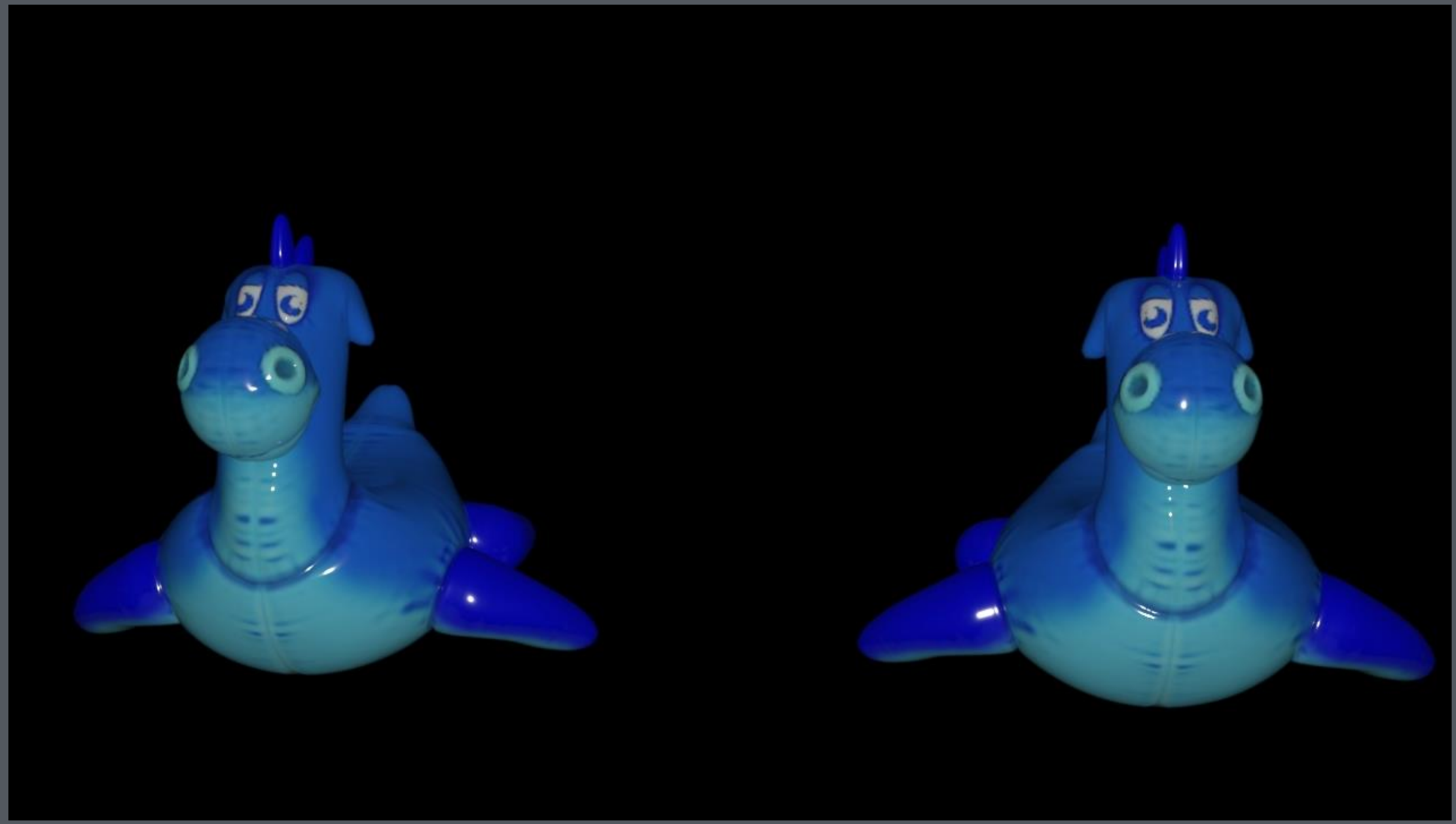
C++ Toronto

A Talk About Small Arrays

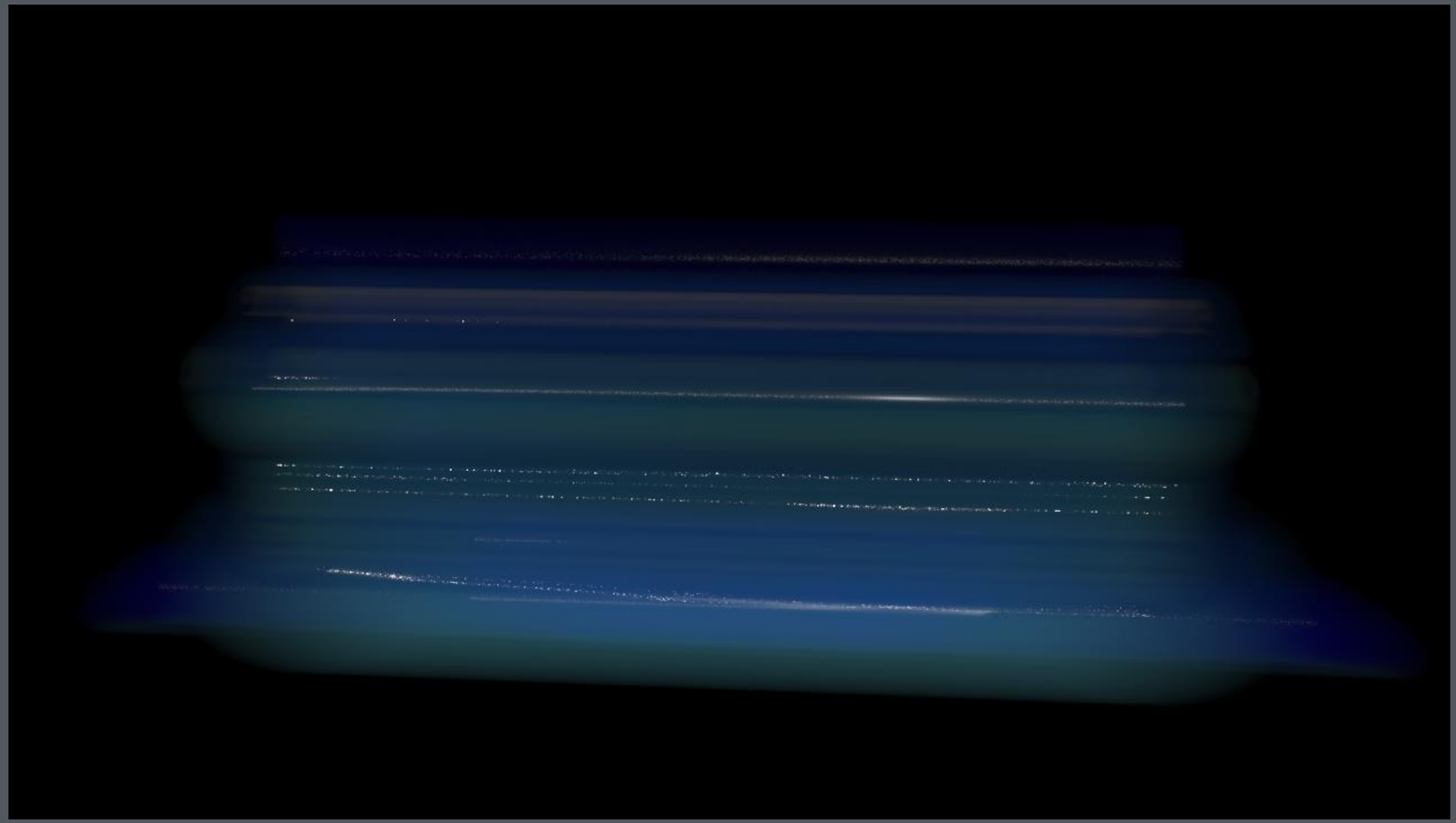
Mark Elendt | SideFX

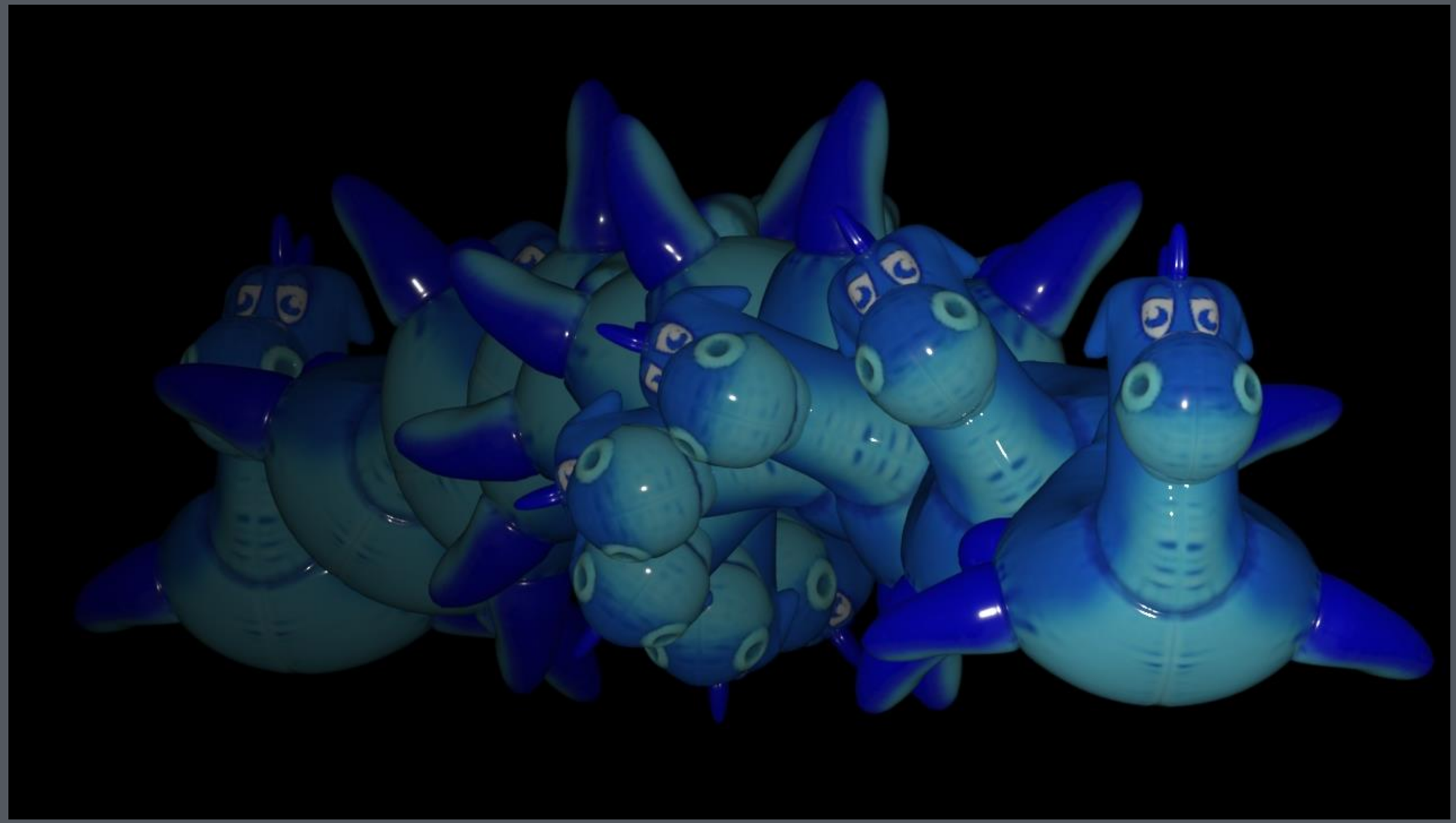
SideFX®

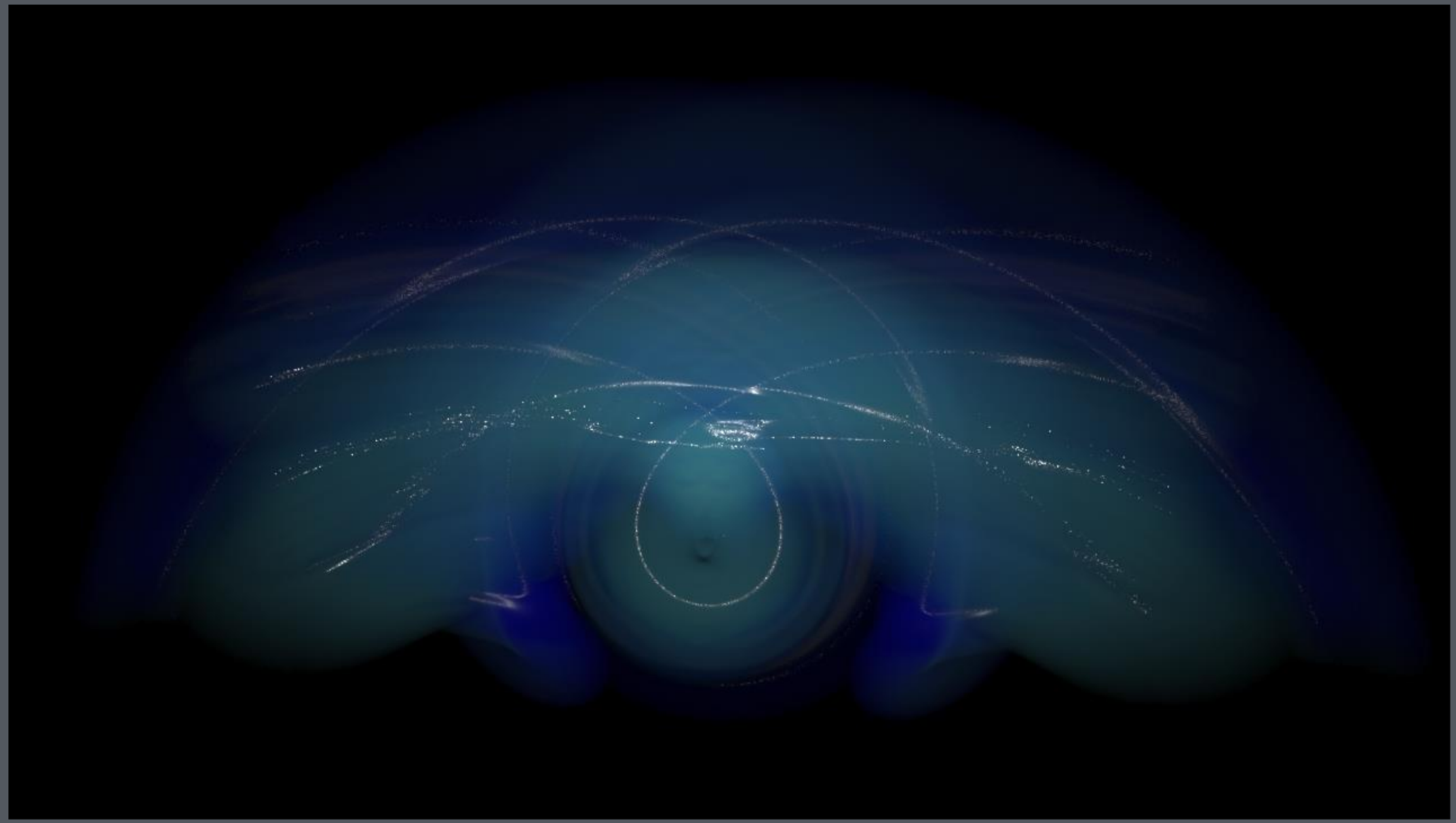




Small Arrays







Fixed Arrays

```
static constexpr int MAX_SEGMENTS = 32;

int
computeTransforms(Mat44 xforms[MAX_SEGMENTS])
{
    double time[MAX_SEGMENTS];
    size_t nsegs = fillMotionTimes(times);
    for (size_t i = 0; i < nsegs; ++i)
       xforms[i] = computeTransform(time[i]);
    return nsegs;
}

void
fillMotionTimes(double times[MAX_SEGMENTS])
{
    assert(_nsegments <= MAX_SEGMENTS);
    for (size_t i = 0; i < _nsegments; ++i)
        times[i] = computeSegmentTime(i, _nsegments));
    return _nsegments;
}
```

Dynamic Arrays

```
void
computeTransforms(std::vector<Mat44> &xforms)
{
    std::vector<double> times;
    fillMotionTimes(times);
    xforms.reserve(times.size());
    for (auto tm : times)
       xforms.push_back(computeTransform(tm));
}

void
fillMotionTimes(std::vector<double> &times)
{
    times.reserve(_nsegments);
    for (size_t i = 0; i < _nsegments; ++i)
        times.push_back(computeSegmentTime(i, _nsegments));
}
```

Dynamic Arrays

```
void
computeTransforms(std::vector<Mat44> &xforms)
{
    std::vector<double> times;
    fillMotionTimes(times);
    xforms.reserve(times.size());
    for (auto tm : times)
       xforms.push_back(computeTransform(tm));
}

void
fillMotionTimes(std::vector<double> &times)
{
    times.reserve(_nsegments);
    for (size_t i = 0; i < _nsegments; ++i)
        times.push_back(computeSegmentTime(i, _nsegments));
}
```

Godbolt - Fixed array

```
void test()
{
#if 0
  std::vector<double> times;
  times.reserve(10);
#else
  double times[10];
#endif
  for (int i = 0; i < 3; ++i)
    times[i] = i;
}
```

```
test():
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-4], 0
.L3:
        cmp     DWORD PTR [rbp-4], 2
        jg      .L4
        cvtsi2sd        xmm0, DWORD PTR [rbp-4]
        mov     eax, DWORD PTR [rbp-4]
        cdqe
        movsd   QWORD PTR [rbp-96+rax*8], xmm0
        add     DWORD PTR [rbp-4], 1
        jmp     .L3
.L4:
        nop
        pop     rbp
        ret
```

```
void test()
{
#if 1
   std::vector<double> times;
   times.reserve(10);
#else
   double times[10];
#endif
   for (int i = 0; i < 3; ++i)
     times[i] = i;
}
```

```
test():
        push    rbp
        mov     rbp, rsp
        push    rbx
        sub     rsp, 40
        lea     rax, [rbp-48]
        mov     rdi, rax
        call    std::vector<double, std::allocator<double> >::vector() [object constructor]
        lea     rax, [rbp-48]
        mov     esi, 10
        mov     rdi, rax
        call    std::vector<double, std::allocator<double> >::reserve(unsigned long)
        mov     DWORD PTR [rbp-20], 0
.L6:
        cmp     DWORD PTR [rbp-20], 2
        jg      .L5
        mov     eax, DWORD PTR [rbp-20]
        movsx   rdx, eax
        lea     rax, [rbp-48]
        mov     rsi, rdx
        mov     rdi, rax
        call    std::vector<double, std::allocator<double> >::operator[](unsigned long)
        cvtsi2sd        xmm0, DWORD PTR [rbp-20]
        movsd   QWORD PTR [rax], xmm0
        add     DWORD PTR [rbp-20], 1
        jmp     .L6
.L5:
        lea     rax, [rbp-48]
        mov     rdi, rax
        call    std::vector<double, std::allocator<double> >::~vector() [object destructor]
        jmp     .L9
        mov     rbx, rax
        lea     rax, [rbp-48]
        mov     rdi, rax
        call    std::vector<double, std::allocator<double> >::~vector() [object destructor]
        mov     rax, rbx
        mov     rdi, rax
        call    _Unwind_Resume
.L9:
        add     rsp, 40
        pop     rbx
        pop     rbp
        ret
```

Godbolt - std::vector

```
void test()
{
#if 1
  std::vector<double> times;
  times.reserve(10);
#else
  double times[10];
#endif
  for (int i = 0; i < 3; ++i)
    times[i] = i;
}
```

```
test():
        push    rbp
        mov     rbp, rsp
        push    rbx
        sub     rsp, 40
        lea     rax, [rbp-48]
        mov     rdi, rax
        call    std::vector<double, std::allocator<double> >::vector() [object constructor]
        lea     rax, [rbp-48]
        mov     esi, 10
        mov     rdi, rax
        call    std::vector<double, std::allocator<double> >::reserve(unsigned long)
        mov     DWORD PTR [rbp-20], 0
.L6:
        cmp     DWORD PTR [rbp-20], 2
        jg      .L5
        mov     eax, DWORD PTR [rbp-20]
        movsx   rdx, eax
        lea     rax, [rbp-48]
        mov     rsi, rdx
        mov     rdi, rax
        call    std::vector<double, std::allocator<double> >::operator[](unsigned long)
        cvtsi2sd        xmm0, DWORD PTR [rbp-20]
        movsd   QWORD PTR [rax], xmm0
        add     DWORD PTR [rbp-20], 1
        jmp     .L6
.L5:
        lea     rax, [rbp-48]
        mov     rdi, rax
        call    std::vector<double, std::allocator<double> >::~vector() [object destructor]
        jmp     .L9
        mov     rbx, rax
        lea     rax, [rbp-48]
        mov     rdi, rax
        call    std::vector<double, std::allocator<double> >::~vector() [object destructor]
        mov     rax, rbx
        mov     rdi, rax
        call    _Unwind_Resume
.L9:
        add     rsp, 40
        pop     rbx
        pop     rbp
        ret
```

Houdini Array Class

Typical implementation of a dynamic array.

```
template <typename T>
class array {
    T *_begin = nullptr;
    T *_end = nullptr
    T *_capacity = nullptr;
public:
    ~array() { delete [] _begin; }
    size_t size() const { return _end - _begin; }
    size_t capacity() const { return _capacity - _begin; }

    void push_back(const T &item) {
        if (capacity() == size())
            reserve(capacity()+BSIZE);
        *_end++ = item;
    }

    void reserve(size_t sz) {
        if (capacity() >= sz) return;
        T *tmp = new T[sz];
        std::copy(_begin, _end, tmp);
        delete [] _begin;
        _end = tmp + size();
        _capacity = tmp + sz;
        _begin = tmp;
    }
};
```

Houdini Small Array

Typical implementation of a dynamic array.

```
template <typename T, size_t STACKSIZE=8>
class small_array : public array<T> {
    T _buffer[STACKSIZE];

public:
    small_array()
    {
        _begin = _buffer;
        _end = _buffer;
        _capacity = _buffer + STACKSIZE;
    }
};
```

Delete non-heap memory!

```
template <typename T>
class array {
    T *_begin = nullptr;
    T *_end = nullptr
    T *_capacity = nullptr;
public:
    ~array() { delete [] _begin; }
    size_t size() const { return _end - _begin; }
    size_t capacity() const { return _capacity - _begin; }

    void push_back(const T &item) {
        if (capacity() == size())
            reserve(capacity()+BSIZE);
        *_end++ = item;
    }

    void reserve(size_t sz) {
        if (capacity() >= sz) return;
        T *tmp = new T[sz];
        std::copy(_begin, _end, tmp);
        delete [] _begin;
        _end = tmp + size();
        _capacity = tmp + sz;
        _begin = tmp;
    }
};
```

Approaches

- std::string
  - SSO
- LLVM
  - separate class

Inheritance

```
class Foo
{
    void  *_begin;
    void  *_end;
    void  *_capacity;
};

class Bar : public Foo
{
    int  _buffer[10];
};


sizeof(Foo)
   = 3 * sizeof(void *)
   = 24 bytes

sizeof(Bar)
   = sizeof(Foo) + 10 * sizeof(int)
   = 24 + 40
   = 64 bytes
```

Inheritance

```
class Foo
{
    void  *_begin;
    void  *_end;
    void  *_capacity;
};

class Bar : public Foo
{
    int  _buffer[10];
};


sizeof(Foo)
   = 3 * sizeof(void *)
   = 24 bytes

sizeof(Bar)
   = sizeof(Foo) + 10 * sizeof(int)
   = 24 + 40
   = 64 bytes
```

void *_begin;
void *_end;
void *_capacity;

void *_begin;
void *_end;
void *_capacity;
int _buffer[10];

Houdini Array Class

Typical implementation of a dynamic array.

```
template <typename T>
class array {
    T *_begin = nullptr;
    T *_end = nullptr
    T *_capacity = nullptr;
public:
    ~array() {
        if (isHeap())
            delete [] _begin;
    }
    size_t size() const { return _end - _begin; }
    size_t capacity() const { return _capacity - _begin; }
    void push_back(const T &item) {
        if (capacity() == size())
            reserve(capacity()+BSIZE);
        *_end++ = item;
    }
    void reserve(size_t sz) {
        if (capacity() >= sz) return;
        T *tmp = new T[sz];
        std::copy(_begin, _end, tmp);
        if (isHeap())
            delete [] _begin;
        _end = tmp + size();
        _capacity = tmp + sz;
        _begin = tmp;
    }
       bool isHeap() const
    {
        return _begin != (T *)((char *)this)+sizeof(*this));
    }
};
```

Function API

Functions all take base array class

```
void
computeTransforms(array<Mat44> &xforms)
{
    array<double> times;
    fillMotionTimes(times);
    xforms.reserve(times.size());
    for (auto tm : times)
       xforms.push_back(computeTransform(tm));
}

void
fillMotionTimes(array<double> &times)
{
    times.reserve(_nsegments);
    for (size_t i = 0; i < _nsegments; ++i)
        times.push_back(computeSegmentTime(i, _nsegments));
}
```

Function API

Functions all take base array class

```
void
computeTransforms(array<Mat44> &xforms)
{
    small_array<double> times;
    fillMotionTimes(times);
    xforms.reserve(times.size());
    for (auto tm : times)
       xforms.push_back(computeTransform(tm));
}

void
fillMotionTimes(array<double> &times)
{
    times.reserve(_nsegments);
    for (size_t i = 0; i < _nsegments; ++i)
        times.push_back(computeSegmentTime(i, _nsegments));
}
```

THANK YOU

Web: SideFX.com

Twitter: sidefx

Facebook: Houdini3D

SideFX®