vscode

Using image manipulation to create post-postmodern art

Jessica Cheng, Kenny Yuan, Anna Lu, Hana Mizuta, Spencer Yen

Remember our LRM?

~~*Lane detection*~~ **ART**

AI Art at Christie's Sells for $432,500

Source: New York Times

Edge detection on images to make modern art:

“Modern art could be achieved *only if* line itself could somehow be prized loose from the task of figuration.”

- *Art critic Michael Fried*

VSCOde Art at Edwards's Sells for $4

We introduce

An Image Manipulation Language

Image as a struct of 3 matrices (R, G, B)
Matrix as an array of array of doubles

Upload images & manipulate underlying matrices

```
func main() -> int {
     image a = load(“photo.jpg”);             // load an image
     image edw = edgedetect(“edwards.jpg”); // or load with built-in filters

     matrix size = dim(“edwards.jpg”);        // and get 1x2 mat of [row, col] size
     matrix r = edw.red;                      // access matrix within image

     matrix m = [1.0, 1.0;…;2.0, 2.0];        // define own matrices
     m = m * r;                               // perform operations on matrices

     print(m[3,0]);                           // access matrix elements
     edw = (m, g, b);                         // store back in image
     save(edw);                               // save image output
     return 0;
}
```

Semant

Semant: Mat to DimMat

Matrix Binop: Need to check matrix sizes for binop

```
| MatLit of expr list list
```

Created a new type that has the row and col size as parameters

```
DimMatrix(rows, cols)
```

MatLit return DimMats, so we only work with DimMats within semant

```
| MatLit m ->
     let rows = (List.length m)
     and cols = (List.length (List.hd m)) in
         if rows = 1 && cols = 0 then DimMatrix(0, 0)
         else DimMatrix(rows, cols)
```

Codegen

Changes to Expr and Stmt in Codegen

Expr

1. Assign
   a. Special for image, matrix (for space allocation)
2. MatLit
3. MatAccess
4. ImageLit
5. ImageRedAccess
6. ImageGreenAccess
7. ImageBlueAccess
8. MatrixRowSize
9. MatrixColSize
10. Call(save)
11. Call(print)

Stmt

1. Local
   a. Mat Local Assignment
   b. Image Local Assign
   c. Built in image functions

Matrix: Storing in Memory

AST DimMatrix Type to LLVM type:

```
DimMatrix (r, c)  -> (array_t (array_t double_t c) r )
```

Dimensions are needed, making matrix size is immutable! This also means we need to know dimensions when we allocate any matrix.

Allocating and storing a MatLit m:

```
L.const_array (array_t double_t (List.length (List.hd m))) array_of_array
```

```
let mat_alloc = L.build_alloca (ltype_of_typ (DimMatrix(row, col))) name builder in
let mat_val = (fst (expr locals_map new_mat_dim_map builder e)) in
L.build_store mat_val mat_alloc builder
```

Note: expr returns a tuple of (L.const_array..., mat_dim_map)

Matrix: Why mat_dim_map?

Consider when we assign a variable to a matrix:

```
matrix m = [1.0, 2.0 ; 3.0, 4.0];
matrix x = m;
matrix y = m + m;
```

To allocate x, we need to know the size of the ID it is being assigned to:

```
let (row, col) = StringMap.find idName mat_dim_map //Get idName row size
in let local_var = L.build_alloca (ltype_of_typ (DimMatrix(row, col)))
                   name builder //Allocate space for x
in let new_locals_map = StringMap.add name local_var locals_map //Add to locals_map
and new_mat_dim_map = StringMap.add name (row, col) mat_dim_map //Add to mat_dim_map
in let matrix_llarray = fst (expr new_locals_map new_mat_dim_map builder e)
in ignore (L.build_store (fst (expr new_locals_map new_mat_dim_map builder e))
local_var builder);
```

Matrix: Accessing Elements

Use build_load on build_gep to retrieve the [row_index, column_index] entry of matrix

```
build_matrix_access m row_index column_index builder locals_map mat_dim_map =
     (try let value = StringMap.find m locals_map in
       (let (r, c) = StringMap.find m mat_dim_map in
         if L.int64_of_const row_index < Some (I.of_int r) // Check if r,c in bounds
          && L.int64_of_const column_index < Some (I.of_int c)
          && L.int64_of_const row_index >= Some (I.of_int 0)
          && L.int64_of_const column_index >= Some (I.of_int 0)
        then L.build_load (L.build_gep (value) [| L.const_int i32_t 0; row_index;
column_index |] m builder) m builder)
        else raise (Failure ("Index out of matrix bounds")))
       with Not_found -> raise (Failure ("Matrix not found: " ^ m))) in
```

Matrix: Binop Sum

Binop allow users to define new filters themselves on top of instant built-in filters

Sum: allocates temp array, loads & stores result in matrix

```
let binop_mat_sum op builder mat_dim_map img_dim_map locals_map m1 m2 m1_row m1_col =
      let new_mat_dim_map = StringMap.add "binop_result" (m1_row, m1_col) mat_dim_map
      and new_mat = L.build_alloca (ltype_of_typ (DimMatrix(m1_row, m1_col))) "binop_result" builder in
      for i=0 to (m1_row - 1) do
        for j=0 to (m1_col - 1) do
          let elem1' = build_matrix_access m1 (L.const_int i32_t i) (L.const_int i32_t j) builder locals_map new_mat_dim_map
          and elem2' = build_matrix_access m2 (L.const_int i32_t i) (L.const_int i32_t j) builder locals_map new_mat_dim_map in
          let result_v = (build_binop_op op) elem1' elem2' "tmp" builder in
          let result_p = L.build_gep new_mat [| L.const_int i32_t 0; L.const_int i32_t i; L.const_int i32_t j |] "" builder in
          ignore(L.build_store result_v result_p builder);
        done
      done;
      ((L.build_load (L.build_gep new_mat [| L.const_int i32_t 0 |] "binop_result" builder) "binop_result" builder),
      (new_mat_dim_map, img_dim_map))
```

Matrix: Binop Multiplication

Multiply elements of m1 row-by-row and m2 col-by-col

Implemented dimension checks with each operation & updated mat_dim_map

```
binop_mat_mult builder mat_dim_map img_dim_map locals_map m1 m2 m1_row m1_col m2_col =
      let new_mat_dim_map = StringMap.add "binop_result" (m1_row, m2_col) mat_dim_map
      and new_mat = L.build_alloca (ltype_of_typ (DimMatrix(m1_row, m2_col))) "binop_result" builder
      and tmp_product = L.build_alloca double_t "tmpproduct" builder in
      ignore(L.build_store (L.const_float double_t 0.0) tmp_product builder);
      for i=0 to (m1_row-1) do
        for j=0 to (m2_col-1) do
          ignore(L.build_store (L.const_float double_t 0.0) tmp_product builder);
          for k=0 to (m1_col-1) do
            let m1_float_val = build_matrix_access m1 (L.const_int i32_t i) (L.const_int i32_t k) builder locals_map new_mat_dim_map
            and m2_float_val = build_matrix_access m2 (L.const_int i32_t k) (L.const_int i32_t j) builder locals_map new_mat_dim_map in
            let product_m1_m2 = L.build_fmul m1_float_val m2_float_val "tmp" builder in
            ignore(L.build_store ( L.build_fadd product_m1_m2 (L.build_load tmp_product "addtmp" builder) "tmp" builder) tmp_product builder);
          done;
          let new_mat_element = L.build_gep new_mat [| L.const_int i32_t 0; L.const_int i32_t i; L.const_int i32_t j |] "tmpmat" builder in
          let tmp_product_val = L.build_load tmp_product "resulttmp" builder in
          ignore(L.build_store tmp_product_val new_mat_element builder);
        done
      done;
      ((L.build_load (L.build_gep new_mat [| L.const_int i32_t 0 |] "binop_result" builder) "binop_result" builder), (new_mat_dim_map, img_dim_map))
in
```

Image type

Image is a struct of an 3 matrices of a set size

```
let image_t = L.named_struct_type context "image_t" in
  L.struct_set_body image_t
    [| (array_t (array_t double_t image_col_size) image_row_size );
       (array_t (array_t double_t image_col_size) image_row_size );
       (array_t (array_t double_t image_col_size) image_row_size )
    |]
  false;
```


Add a level of indirection
Store everything on the stack

Image type

Set image sizes because matrices are implemented as array types
Array types take in the row and col size as parameters

```
let image_row_size = 50
and image_col_size = 50 in
```

Originally wanted a 4096 x 2160 size image, but took too long to compile because images are on the stack

50 x 50 pixel images

Image Functions

Accessing red, blue and green matrices of image

```
ImageRedAccess img_id  ->
    let img_val = StringMap.find img_id locals_map in
      let pointer_to_red = L.build_struct_gep img_val 0 "" builder in
      ((L.build_load pointer_to_red "actual_red" builder), (mat_dim_map,
         img_dim_map))
```

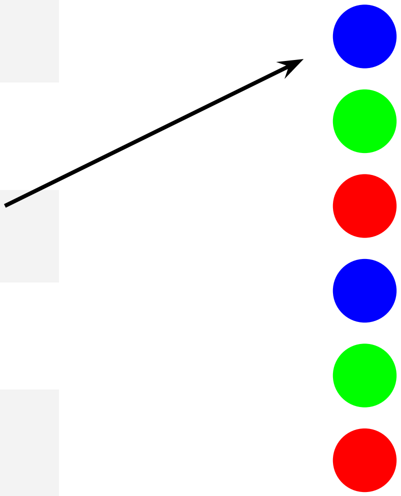
Load (OpenCV via C++)

Input:

```
Mat img = imread(imageName, CV_LOAD_IMAGE_COLOR);
```

Output:

```
double* output = new double[3 * height * width];
```



Implementation:

```
b = input[img.step * i + j * img.channels()];
output[k++] = b;
...
```

Load (OCaml)

Allocation:

```
let img = L.build_alloca (ltype_of_typ (Image)) "img" builder in
let r_ptr = L.build_struct_gep img 0 "r_ptr" builder
and g_ptr = L.build_struct_gep img 1 "g_ptr" builder
and b_ptr = L.build_struct_gep img 2 "b_ptr" builder in
```

Load (OCaml)

Get each element:

```
L.build_load (L.build_gep load_return [| L.const_int i32_t i |]
"tmp" builder) "tmp" builder
```

Get pointer to allocated space:

```
L.build_gep r_ptr [| L.const_int i32_t 0; L.const_int i32_t row;
L.const_int i32_t col |] "ptr" builder in
```

Store element in allocated space:

```
ignore(L.build_store next_element red_elem_ptr builder); )
```

Save (OCaml)

Allocate space for temp image & store image value:

```
let img_struct_alloc = L.build_alloca image_t "tmp_img_alloc" builder in
let img_struct_val = fst (expr locals_map mat_dim_map img_dim_map builder e) in
L.build_store img_struct_val img_struct_alloc builder
```

Get pointers to temp im matrices:

```
let pointer_to_red = L.build_struct_gep img_struct_alloc 0 "i_red" builder in
let _  = L.build_load pointer_to_red "actual_red" builder in
```

Pass double[r][c] to cpp, where it populates double[ ] & writes to image file

```
let ptr_typ = L.pointer_type (array_t double_t image_row_size) in
let save_cpp_t = L.function_type void_t [| ptr_typ; ptr_typ; ptr_typ |] in
L.build_call save_cpp_func [| red_mat_ptr; green_mat_ptr; blue_mat_ptr |] ""
builder
```

Other Built-In Functions

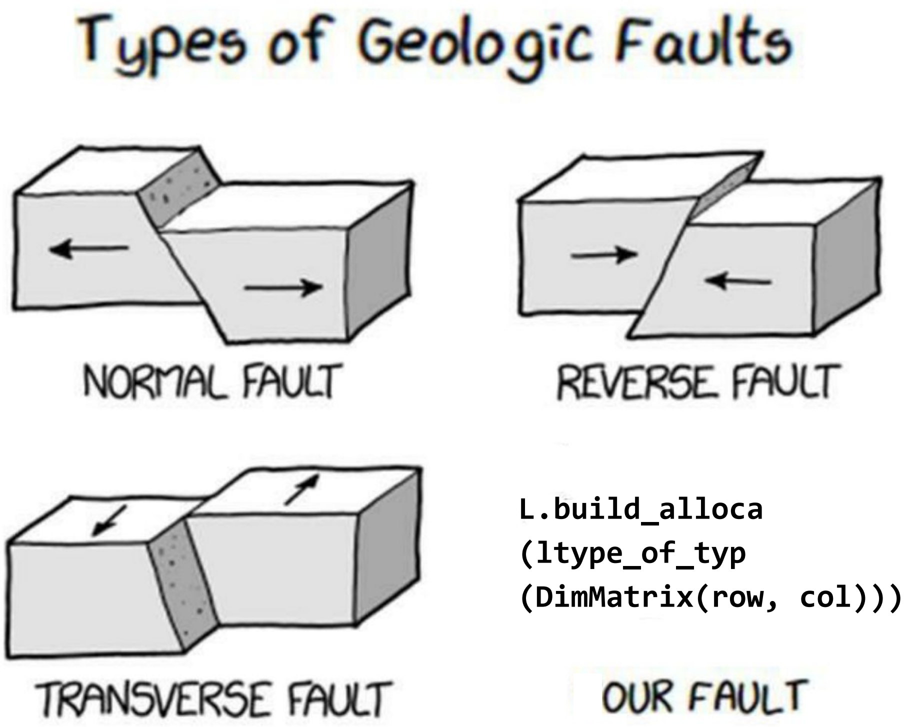
```
load(string s) -> image
save(matrix m) -> void

print( ____ ) -> void

blur(string s) -> image
brighten(string s) -> image
grayscale(string s) -> image
edgedetect(string s) -> image
```

Other Built-In Functions

```
stephenedwards(string s) -> image
```


Types of Geologic Faults
NORMAL FAULT
REVERSE FAULT
TRANSVERSE FAULT
L.build_alloca
(ltype_of_typ
(DimMatrix(row, col)))
OUR FAULT

100x100 pixel image

10,000 elements in array array

100,000+ lines of gep, load, store in IR

```
%element_ptr89970 = getelementptr double, double* %edgedetect_ret, i32 29991
%val89971 = load double, double* %element_ptr89970
%green_mat_ptr89972 = getelementptr [100 x [100 x double]], [100 x [100 x double]]* %b_ptr,
store double %val89971, double* %green_mat_ptr89972
%element_ptr89973 = getelementptr double, double* %edgedetect_ret, i32 29992
%val89974 = load double, double* %element_ptr89973
%green_mat_ptr89975 = getelementptr [100 x [100 x double]], [100 x [100 x double]]* %g_ptr,
store double %val89974, double* %green_mat_ptr89975
%element_ptr89976 = getelementptr double, double* %edgedetect_ret, i32 29993
%val89977 = load double, double* %element_ptr89976
%red_mat_ptr89978 = getelementptr [100 x [100 x double]], [100 x [100 x double]]* %r_ptr, i3
store double %val89977, double* %red_mat_ptr89978
%element_ptr89979 = getelementptr double, double* %edgedetect_ret, i32 29994
%val89980 = load double, double* %element_ptr89979
%green_mat_ptr89981 = getelementptr [100 x [100 x double]], [100 x [100 x double]]* %b_ptr,
store double %val89980, double* %green_mat_ptr89981
%element_ptr89982 = getelementptr double, double* %edgedetect_ret, i32 29995
%val89983 = load double, double* %element_ptr89982
%green_mat_ptr89984 = getelementptr [100 x [100 x double]], [100 x [100 x double]]* %g_ptr,
store double %val89983, double* %green_mat_ptr89984
%element_ptr89985 = getelementptr double, double* %edgedetect_ret, i32 29996
%val89986 = load double, double* %element_ptr89985
%red_mat_ptr89987 = getelementptr [100 x [100 x double]], [100 x [100 x double]]* %r_ptr, i3
store double %val89986, double* %red_mat_ptr89987
%element_ptr89988 = getelementptr double, double* %edgedetect_ret, i32 29997
%val89989 = load double, double* %element_ptr89988
%green_mat_ptr89990 = getelementptr [100 x [100 x double]], [100 x [100 x double]]* %b_ptr,
store double %val89989, double* %green_mat_ptr89990
%element_ptr89991 = getelementptr double, double* %edgedetect_ret, i32 29998
%val89992 = load double, double* %element_ptr89991
%green_mat_ptr89993 = getelementptr [100 x [100 x double]], [100 x [100 x double]]* %g_ptr,
store double %val89992, double* %green_mat_ptr89993
%element_ptr89994 = getelementptr double, double* %edgedetect_ret, i32 29999
%val89995 = load double, double* %element_ptr89994
%red_mat_ptr89996 = getelementptr [100 x [100 x double]], [100 x [100 x double]]* %r_ptr, i3
store double %val89995, double* %red_mat_ptr89996
```

What We Should've Done With:

```
L.build_alloca (ltype_of_typ (DimMatrix(row, col)))
```

1. Allocate DimMatrix on the heap (array array with |row*col| elements)
2. Use pointers to doubles, so we don't have to reallocate and store all elements any time matrix is passed (binop)

As we implemented matrix, we only tested on small matrices.

Once we got to image, we realized this was a problem.

What's Next?

1. Store everything on the heap instead of the stack
2. Implement matrix elements as pointers: saves memory, space and makes matrices mutable
3. Write more filter-related features directly into OCaml
4. Variable image dimensions

Demo