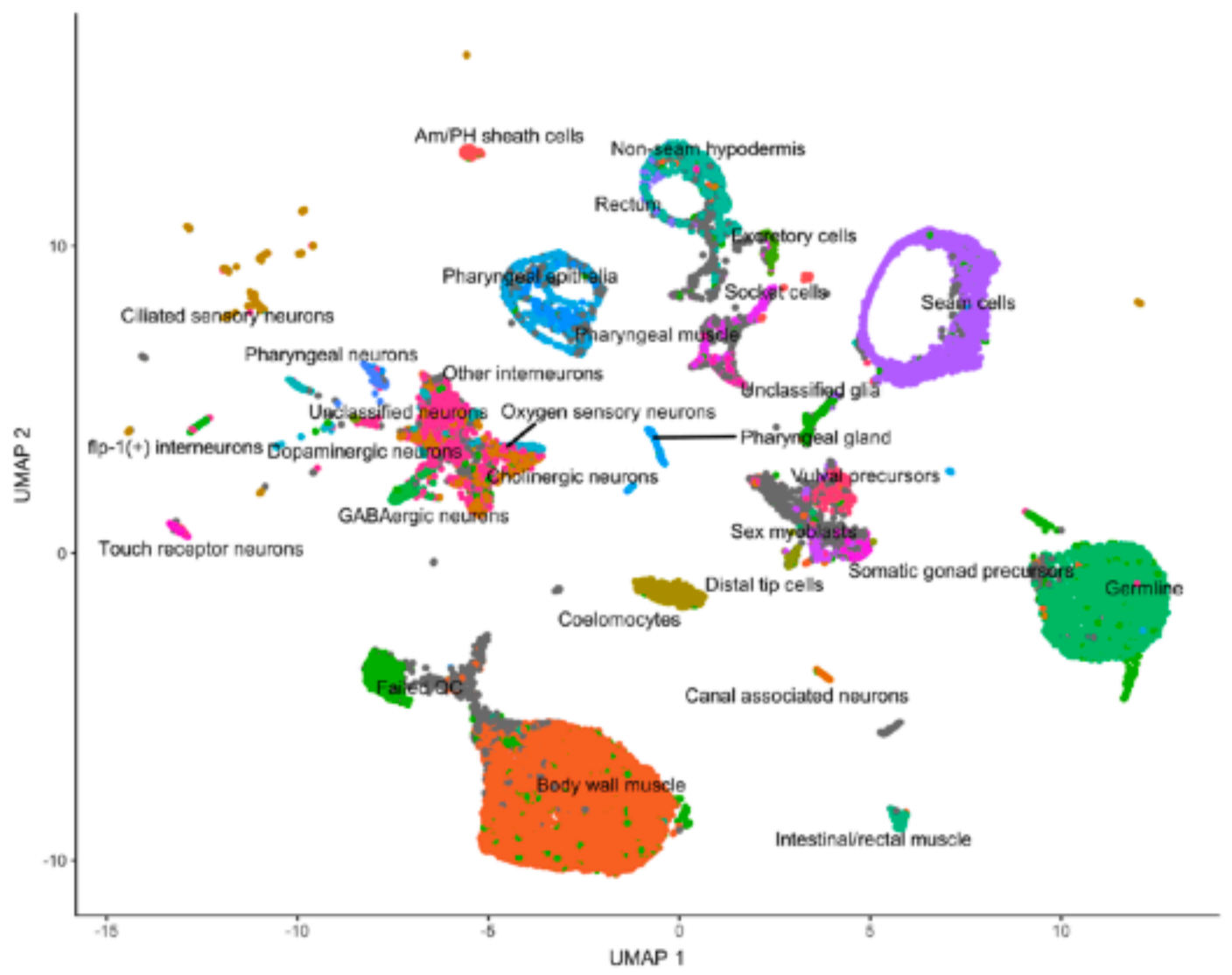
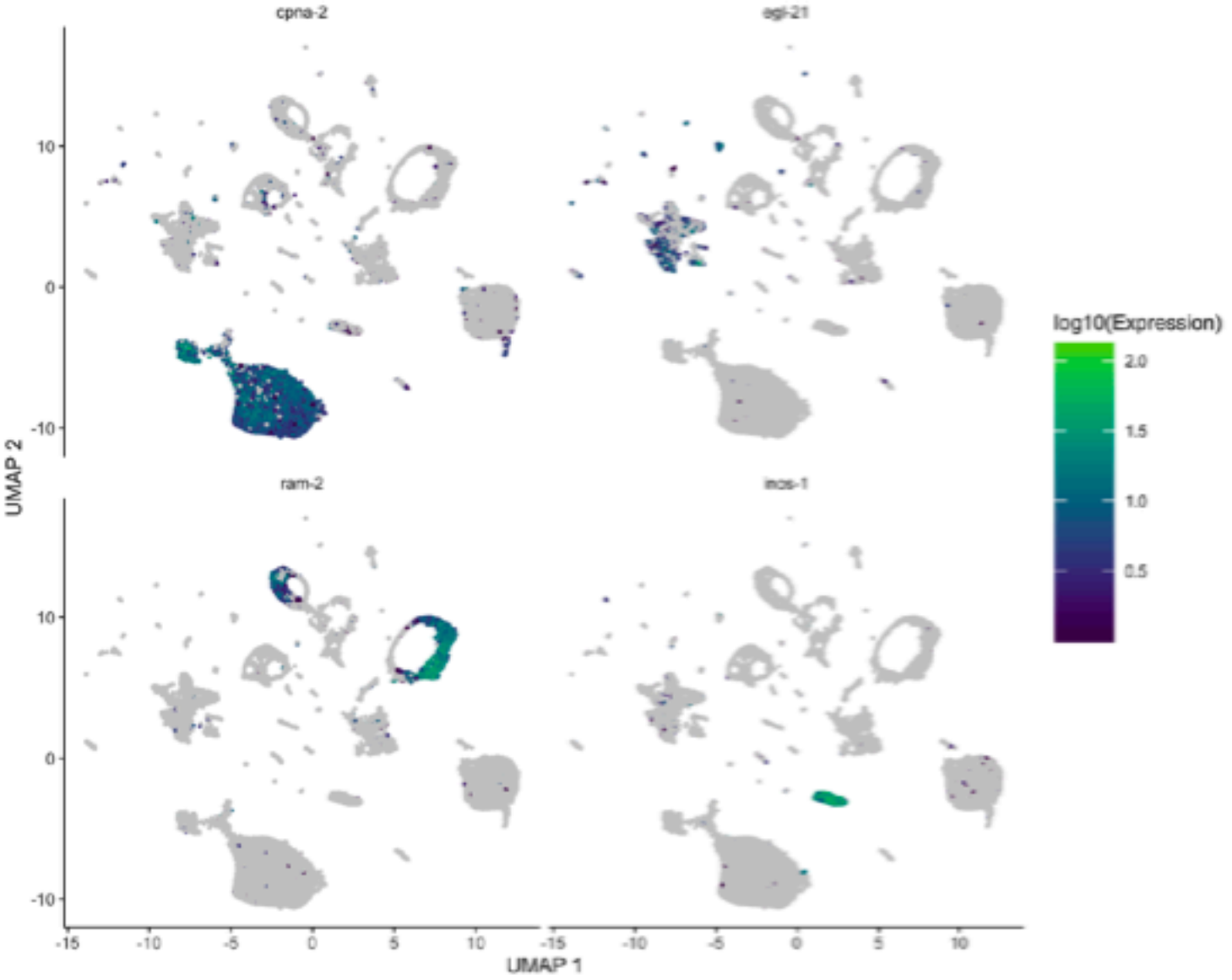
# Plots can be built programmatically

```
plot_cells <- function(cds,
                       x=1,
                       y=2,
                       reduction_method = c("UMAP", "tSNE", "PCA", "LSI", "Aligned"),
                       color_cells_by="cluster",
                       group_cells_by="cluster", "partition"),
                       genes=NULL,
                       show_trajectory_graph=TRUE,
                       trajectory_graph_segment_size=0.75,
                       trajectory_graph_color="grey28",
                       norm_method = c("log", "size_only"),
                       label_cell_groups = TRUE,
                       label_groups_by_cluster=TRUE,
                       label_leaves=FALSE,
                       label_branch_points=FALSE,
                       label_roots=FALSE,
                       label_leaves=FALSE,
                       label_nodes=FALSE,
                       cell_size=1,
                       cell_stroke I(cell_size / 2),
                       alpha=0.5,
                       min_expr=0.1,
                       rasterize=FALSE,
                       scale_to_range=FALSE) {
reduction_method <- match.arg(reduction_method)
assert_cds(cds, TRUE, "Not a valid cell_data_set")
assert_char(x)[is.null(reduced_dim(cds))[reduction_method]],
            msg = paste("No dimensionality reduction for",
                        reduction_method,
                        "Please run reduce_dimension with",
                        "reduction_method =", reduction_method,
                        "before attempting to plot.")
low_dim_coords <- reduced_dims(cds)[[reduction_method]]
assert_char(colnames(low_dim_coords)[c(x,y)],
            msg = paste("Cannot find", x, "and", y, "in", reduction_method,
                        "be dimensions in reduced dimension",
                        "space.")

if(!is.null(color_cells_by) {
  assert_char(c(color_cells_by %in% c("cluster", "partition",
                                      "pseudotime") |
                color_cells_by %in% names(colData(cds)),
                msg = paste("Error: color_cells_by must",
                            "one of 'cluster', 'partition', 'pseudotime', ",
                            "or a column in the colData table.")

  if(color_cells_by == "pseudotime") {
    trajectory_names(cds, reduction_method = reduction_method),
            error = function(e) {
              stop(paste("No pseudotime for", reduction_method,
                         "trajectories. Please run order_cells()",
                         "with reduction_method =",
                         "before attempting to color by pseudotime."))

  }
}
assert_char=assert_char(!is.null(color_cells_by) || !is.null(markers),
            msg = paste("Either Color_cells_by or markers must",
                        "be NULL, cannot color by both!!")


norm_method = match.arg(norm_method)
group_cells_by=match.arg(group_cells_by)
assert_char=assert_char(!is.null(color_cells_by) || !is.null(genes),
                        msg = paste("Either color_cells_by or genes must be",
                                    "NULL, cannot color by both!!")


if (show_trajectory_graph &&
  is.null(principal_graph(cds)[[reduction_method])) {
  warning("Cannot find graph, Has learn_graph() been called yet?")
  show_trajectory_graph = FALSE
}

norm_method <- NA
sample_name <- NA
assay_state <- colData(cds)$State
data_dim_1 <- NA
data_dim_2 <- NA
if (TestReal){
  plotting_func <- ggrastr::geom_point_rast
}else{
  plotting_func <- ggplot2::geom_point
}

S_matrix <- reduced_dims(cds)[[reduction_method]]
data_df <- data.frame(S_matrix[,c(x,y)])

colnames(data_df) <- c("data_dim_1", "data_dim_2")
data_df$sample_name <- row.names(data_df)

data_df <- as.data.frame(cbind(data_df, colData(cds)))
if (group_cells_by == "cluster"){
  data_df$cell_group <-
    tryCatch(clusters(cds,
                      reduction_method = reduction_method)[
                        data_df$sample_name],
             error = function(e) {NULL})
} else if (group_cells_by == "partition") {
  data_df$cell_group <-
    tryCatch(partitions(cds,
                        reduction_method = reduction_method)[
                          data_df$sample_name],
             error = function(e) {NULL})
} else{
  stop("Error: unrecognized way of grouping cells.")
}

if (color_cells_by == "cluster"){
  data_df$cell_color <-
    tryCatch(clusters(cds,
                      reduction_method = reduction_method)[
                        data_df$sample_name],
             error = function(e) {NULL})
} else if (!color_cells_by == "partition") {
  data_df$cell_color <-
    tryCatch(partitions(cds,
                        reduction_method = reduction_method)[
                          data_df$sample_name],
             error = function(e) {NULL})
} else if (color_cells_by == "pseudotime") {
  data_df$cell_color <-
    tryCatch(pseudotime(cds,
                        reduction_method = reduction_method)[
                          data_df$sample_name]), error = function(e) {NULL})
} else{
  data_df$cell_color <- colData(cds)[data_df$sample_name,color_cells_by]
}

## Graph info
if (show_trajectory_graph) {

  ica_space_df <- t(ica_space_df[principal_graph_aux[[reduction_method]]$dp_mst) %>%
    as.data.frame() %>%
    dplyr::select_(prin_graph_dim_1 = x, prin_graph_dim_2 = y) %>%
    dplyr::mutate(sample_name = rownames(.),
                  sample_state = rownames(.))

  dp_mst <- cds[principal_graph[[reduction_method]]]

  edge_df <- dp_mst %>%
    igraph::as_data_frame() %>%
    dplyr::select(source = "from", target = "to") %>%
    dplyr::left_join(ica_space_df %>%
                     dplyr::select(.,
                                   source = sample_name,
                                   source_prin_graph_dim_1="prin_graph_dim_1",
                                   source_prin_graph_dim_2="prin_graph_dim_2"),
                     by = "source") %>%
    dplyr::left_join(ica_space_df %>%
                     dplyr::select(.,
                                   target="sample_name",
                                   target_prin_graph_dim_1="prin_graph_dim_1",
                                   target_prin_graph_dim_2="prin_graph_dim_2"),
                     by = "target")
}

## Marker genes
markers_exprs <- NULL
expression_legend_label <- NULL
if (!is.null(genes)){
  if (!is.null(dim(genes)) && dim(genes) >= 2){
    stop("genes must be a vector, a 1 dim matrix or a 1 col matrix!")
  } else {
    markers = genes
  }
  markers_rowData <- as.data.frame(subset(rowData(cds),
                                          gene_short_name %in% markers |
                                          row.names(rowData(cds)) %in%
                                          markers))

  if (nrow(markers_rowData) == 0) {
    stop("None of the provided genes were found in the cds")
  }
  if (nrow(markers_rowData) >= 1) {
    cds_exprs <- SingleCellExperiment::counts(cds)[row.names(markers_rowData), ,drop=FALSE]
    cds_exprs <- Matrix::t(Matrix::t(cds_exprs)/size_factors(cds))

    if (!is.null(dim(genes)) && dim(genes) >= 2){
    # if (!is.null(dim(genes)) && ncol(genes) >= 2){
    row.names(genes) = genes[,1]
    genes = genes[row.names(cds_exprs),]
    }

    agg_mat = as.matrix(aggregate_gene_expression(cds, genes, norm_method=norm_method, scale_agg_values=FALSE))
    agg_mat = agg_mat[ncol(agg_mat):1]
    markers_exprs <- reshape2::melt(markers_exprs)
    colnames(markers_exprs)[1:2] <- c('feature_id', 'cell_id')
    if (is.factor(genes[,2])){
      markers_exprs$feature_id = factor(markers_exprs$feature_id,
                                        levels=levels(genes[,2]))
    }

    markers_exprs$feature_label <- markers_exprs$feature_id
    if (!is.null(genes)){
    expression_legend_label = "Expression score"
    } else {
      cds_exprs$g = round(10000*cds_exprs$g)/10000
      markers_exprs = matrix(cds_exprs, ncol=ncol(markers_rowData))
      row.names(markers_exprs) = row.names(SingleCellExperiment::counts(cds))
      row.names(markers_exprs) = row.names(markers_rowData)
      markers_exprs <- reshape2::melt(markers_exprs)
      colnames(markers_exprs)[1:2] <- c('feature_id', 'cell_id')
      markers_exprs <- merge(markers_exprs, markers_rowData,
                             by.x="feature_id", by.y="row.names")
      if (is.null(markers_exprs$gene_short_name)){
        markers_exprs$feature_label <-
          as.character(markers_exprs$feature_id)
      } else {
        markers_exprs$feature_label <-
          as.character(markers_exprs$gene_short_name)
      }

      markers_exprs$feature_label <- ifelse(is.na(markers_exprs$feature_label) | as.character(markers_exprs$feature_label) %in% markers,
                                            as.character(markers_exprs$feature_id),
                                            as.character(markers_exprs$feature_label))

      markers_exprs$feature_label <- factor(markers_exprs$feature_label,
                                            levels = markers)

      if (norm_method == "size_only")
      expression_legend_label = "Expression"
      else
      expression_legend_label = "log10(Expression)"
    }

    if (scale_to_range){
      markers_exprs = dplyr::group_by(markers_exprs, feature_label) %>%
      dplyr::mutate(max_val_for_feature = max(value),
                    min_val_for_feature = min(value)) %>%
      dplyr::mutate(value = 100 * (value - min_val_for_feature) / (max_val_for_feature - min_val_for_feature))
      expression_legend_label = "% Max"
    }
  }

if (!label_cell_groups && is.null(color_cells_by) == FALSE){
  if (!is.logical(color_cells[1])){
    if (!is.null(legend.position.
    message(paste(color_cells_by, "not found in colData(cds), cells will",
                  "not be colored"))
  }
  text_df = NULL
  label_cell_groups = FALSE
}else{
  if(!is.character(data_df$cell_color) || is.factor(data_df$cell_color)) {
    if (label_groups_by_cluster && is.null(data_df$cell_group) == FALSE){
      text_df = data_df %>%
        dplyr::group_by(cell_group) %>%
        dplyr::mutate(cell_group = median(data_dim_1)) %>%
        dplyr::group_by(cell_color, cell_group=TRUE) %>%
        dplyr::summarize(prop=length(cell_group)/cells_in_cluster)
      median_coord_df = text_df %>%
        dplyr::summarize(fraction_of_group = dplyr::n(),
                         text_x = stats::median(x = data_dim_1),
                         text_y = stats::median(x = data_dim_2))
      text_df = suppressMessages(text_df %>% dplyr::select(cell_group) %>%
                                 dplyr::inner_join(median_coord_df))
      text_df = suppressMessages(dplyr::inner_join(text_df,
                                                   median_coord_df))
      text_df = text_df %>% dplyr::group_by(cell_group) %>%
      dplyr::mutate(perc=1)
      text_df = data_df %>% dplyr::group_by(cell_color) %>%
        dplyr::mutate(perc=1)
      median_coord_df = text_df %>%
        dplyr::summarize(fraction_of_group = dplyr::n(),
                         text_x = stats::median(x = data_dim_1),
                         text_y = stats::median(x = data_dim_2))
      text_df = suppressMessages(text_df %>% dplyr::select(perc) %>%
                                 dplyr::inner_join(median_coord_df))
      text_df = suppressMessages(dplyr::inner_join(text_df,
                                                   median_coord_df))
      text_df = text_df %>% dplyr::group_by(cell_color) %>%
        dplyr::top_n(1,wt=perc)
    }
    text_df(label = as.character(text_df %>% dplyr::pull(cell_color))
    # Turn factors into strings, if they are
    # factors and the factor is not a good reason for the bit below, but i
    # don't want to deal with it now
    # text_df$label <- paste0(1:length(text_df))
    # text_df$process_label <- paste0(1:length(text_df), '_')
    # as.character(as.matrix(text_df[, 1])[
    # text_df$process_label <- NULL
    # name(ccs(process_label) <- as.character(as.matrix(text_df[, 1]))
    # data_df[, group_by] <-
    # process_labels[as.character(data_df[, group_by])]
    # text_df$label = process_label
    } else {
      message(paste("Cells aren't colored in a way that allows them to",
                    "be grouped."))
    text_df = NULL
    label_cell_groups = FALSE
  }
}
if (!is.null(markers_exprs) && nrow(markers_exprs) > 0){
  data_df <- merge(data_df, markers_exprs, by.x="sample_name",
                   by.y="cell_id")
  data_df$value <- ifelse(data_df$ value >= min_expr, value, NA))
na_sub <- data_df[is.na(data_df$value),]
if(score_method == "size_only"){
  ~ ggplot(data=data_df, aes(x=data_dim_1, y=data_dim_2))
  plotting_func(aes(color=NA), data=na_sub, size=I(cell_size),
                stroke = I(cell_stroke), color = "grey28", alpha = alpha,
                data = na_sub) +
  plotting_func(aes(color=value), size=I(cell_size),
                stroke = I(cell_stroke), na.rm = TRUE,
  viridis::scale_color_viridis(option = "viridis",
                               name = expression_legend_label,
                               na.value = "grey50", end = 0.8,
                               alpha = alpha)
    guides(alpha = FALSE) + facet_grap(~feature_label)
  } else {
  g <- ggplot(data=data_df, aes(x=data_dim_1, y=data_dim_2)) +
  plotting_func(aes(data_dim_1, data_dim_2), size=I(cell_size),
                stroke = I(cell_stroke), color = "grey28",
                data = na_sub, alpha = alpha) +
  plotting_func(aes(color=log10(value+min_expr)),
                size=I(cell_size), stroke = I(cell_stroke),
                na.rm = TRUE, alpha = alpha)
  viridis::scale_color_viridis(option = "viridis",
                               name = expression_legend_label,
                               na.value = "grey50", end = 0.8,
                               alpha = alpha)
    guides(alpha = FALSE) + facet_grap(~feature_label)
  }
} else {
  g <- ggplot(data=data_df, aes(x=data_dim_1, y=data_dim_2))

# We don't want to force users to call order_cells before even being able
# to look at the trajectory, so check whether it's null and if so, just
# ignore the error:
if(color_cells_by == "cluster", "partition"){
  if (!is.null(data_df$cell_color)){
    g <- g + geom_point(color="grey", size=I(cell_size),
                        stroke = I(cell_stroke), na.rm = TRUE,
                        alpha = alpha)
  message(paste("cluster_cells() has not been called yet, can't",
                "color cells by cluster"))
  } else{
    g <- g + geom_point(aes(color = cell_color), size=I(cell_size),
                        stroke = I(cell_stroke), na.rm = TRUE,
                        alpha = alpha)
    g <- g + guides(color = guide_legend(title = color_cells_by,
                                         override.aes = list(size = 4)))
  } else if (class(data_df$cell_color) == "numeric"){
    g <- g + geom_point(aes(color = cell_color), size=I(cell_size),
                        stroke = I(cell_stroke), na.rm = TRUE, alpha = alpha)
    g <- g + viridis::scale_color_viridis(name = color_cells_by, option="C")
  } else {
    g <- g + geom_point(aes(color = cell_color), size=I(cell_size),
                        stroke = I(cell_stroke), na.rm = TRUE, alpha = alpha)
    g <- g + guides(color = guide_legend(title = color_cells_by,
                                         override.aes = list(size = 4)))
  }
}
if (show_trajectory_graph){
  g <- g + geom_segment(aes_string(x="source_prin_graph_dim_1",
                                   y="source_prin_graph_dim_2",
                                   xend="target_prin_graph_dim_1",
                                   yend="target_prin_graph_dim_2",
                        size=trajectory_graph_segment_size,
                        color=I(trajectory_graph_color),
                        linetype="solid",
                        na.rm=TRUE,
                        data=edge_df)
                        )


  if (label_branch_points){
    mst_branch_nodes <- branch_nodes(cds)
    branch_points_df <- ica_space_df %>%
      dplyr::slice(match(names(mst_branch_nodes), sample_name)) %>%
      dplyr::mutate(branch_point_idx = seq_len(ncol(.)[,1]))

    g <- g +
    geom_point(aes_string(x="prin_graph_dim_1", y="prin_graph_dim_2"),
               shape = 21, stroke=I(trajectory_graph_segment_size),
               color="black",
               fill="white",
               size=I(graph_label_size * 1.5),
               na.rm=TRUE,
    geom_text(aes_string(x="prin_graph_dim_1", y="prin_graph_dim_2",
                         label="branch_point_idx"),
              size=I(graph_label_size), color="white", na.rm=TRUE,
              branch_point_df)
              )


  if (label_leaves){
    mst_leaf_nodes <- leaf_nodes(cds)
    leaf_df <- ica_space_df %>%
      dplyr::slice(match(names(mst_leaf_nodes), sample_name)) %>%
      dplyr::mutate(leaf_idx = seq_len(ncol(.)[,1]))

    g <- g +
    geom_point(aes_string(x="prin_graph_dim_1", y="prin_graph_dim_2"),
               shape = 21, stroke=I(trajectory_graph_segment_size),
               color="black",
               fill="white",
               size=I(graph_label_size * 1.5),
               na.rm=TRUE,
    geom_text(aes_string(x="prin_graph_dim_1", y="prin_graph_dim_2",
                         label="leaf_idx"),
              size=I(graph_label_size), color="black", na.rm=TRUE, leaf_df)
              )

  if (label_roots){
    mst_root_nodes <- root_nodes(cds)
    root_df <- ica_space_df %>%
      dplyr::slice(match(names(mst_root_nodes), sample_name)) %>%
      dplyr::mutate(root_idx = seq_len(ncol(.)[,1]))

    g <- g +
    geom_point(aes_string(x="prin_graph_dim_1", y="prin_graph_dim_2"),
               shape = 21, stroke=I(trajectory_graph_segment_size),
               color="black",
               fill="white",
               size=I(graph_label_size * 1.5),
               na.rm=TRUE,
    geom_text(aes_string(x="prin_graph_dim_1", y="prin_graph_dim_2",
                         label="root_idx"),
              size=I(graph_label_size), color="black", na.rm=TRUE, root_df)
              )
}


if(label_cell_groups){
  g <- g + ggrepel::geom_text_repel(data = text_df,
                                    mapping = aes_string(x = "text_x",
                                                         y = "text_y",
                                                         label = "label"),
                                    size=I(graph_label_size)
  if (is.null(markers_exprs)){
    g <- g + theme(legend.position="none")
}

g <- g +
#scale_color_brewer(palette="Set2") +
ggplot2::theme_bw() +
ggplot2::theme(panel.gridmajor = NULL,
ggplot2::theme(panel.gridminor = NULL,
ggplot2::theme(legend.key = element_blank()) +
ggplot2::theme(legend.position = "top") +
theme(legend.key.height=unit(0.5,"line"),
theme(panel.background = element_rect(fill="white"))

g
}
```

```r
if(label_cell_groups) {
    g <- g + ggrepel::geom_text_repel(data = text_df,
                                      mapping = aes_string(x = "text_x",
                                                           y = "text_y",
                                                           label = "label"),
                                      size=I(group_label_size))
    # If we're coloring by gene expression, don't hide the legend
    if (is.null(markers_exprs))
      g <- g + theme(legend.position="none")
  }
```


Am/PH sheath cells
Non-seam hypodermis
Rectum
Excretory cells
Pharyngeal epithelia
Socket cells
Seam cells
Ciliated sensory neurons
Pharyngeal muscle
Pharyngeal neurons
Other interneurons
Unclassified glia
Unclassified neurons
Oxygen sensory neurons
Pharyngeal gland
flp-1(+) interneurons
Dopaminergic neurons
Cholinergic neurons
Vulval precursors
GABAergic neurons
Sex myoblasts
Touch receptor neurons
Somatic gonad precursors
Distal tip cells
Germline
Coelomocytes
Failed QC
Canal associated neurons
Body wall muscle
Intestinal/rectal muscle
UMAP 2
UMAP 1
10
0
-10
-15
-10
-5
0
5
10

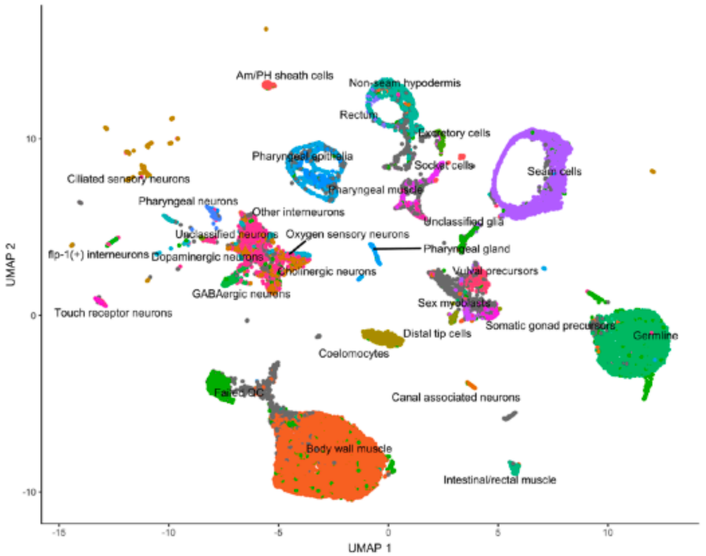

cpna-2
agl-21
ram-2
inos-1
UMAP 2
UMAP 1
10
0
-10
-15
-10
-5
0
5
10
log10(Expression)
2.0
1.5
1.0
0.5

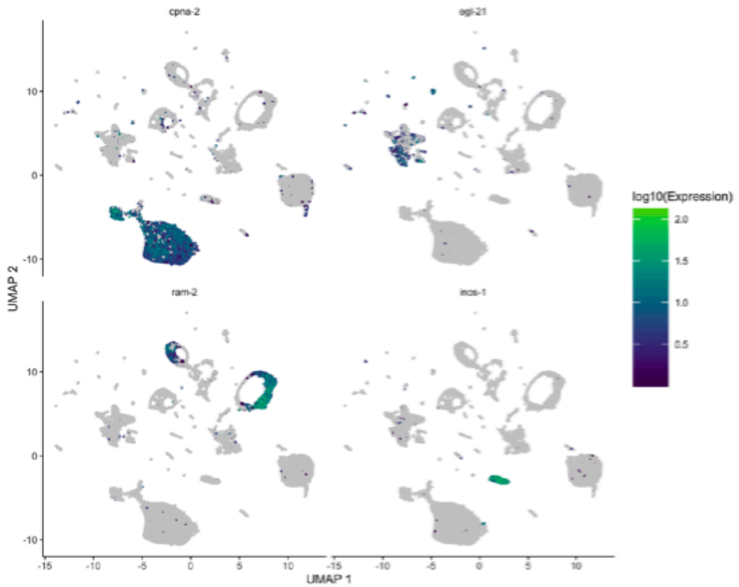
`plot_cells()`

from Monocle 3:

# Plots can be built programmatically

`plot_cells()`
from Monocle 3:




```
if(label_cell_groups) {
    g <- g + ggrepel::geom_text_repel(data = text_df,
                                      mapping = aes_string(x = "text_x",
                                                           y = "text_y",
                                                           label = "label"),
                                      size=I(group_label_size))
    # If we're coloring by gene expression, don't hide the legend
    if (is.null(markers_exprs))
      g <- g + theme(legend.position="none")
  }
```

# THE END

For now.